Parallel Computer Architectures for the SPMD Programming Model

博士学位論文

論文題目　Parallel Computer Architectures for the SPMD Programming Model
(SPMD 型並列プログラミングモデルのための並列計算機アーキテクチャに関する研究)

提出者　東北大学大学院情報科学研究科
情報基礎科学 専攻

学籍番号　7DS3

氏名　北島宏之

指導教官	中村維男　教授
審査委員 （○印は主査）	○ 中村維男　教授 1 根元義章　教授　2 静谷啓樹　教授 3 小林広明　助教授　4 　　教授 5 　　教授　6 　　教授

Doctoral Dissertation

Parallel Computer Architectures for the SPMD Programming Model

(SPMD型並列プログラミングモデルのための並列計算機アーキテクチャに関する研究)

Hiroyuki Kitajima

Department of Computer and Mathematical Sciences,
Graduate School of Information Sciences,
Tohoku University

January 20, 2000

Contents

List of Figures

List of Tables

Acknowledgements

It is not able to complete this research without the strong support and kindly help from my supervisors, my colleagues, and my family. Therefore, I would like to thank the people who have directed and supported me to accomplish this thesis.

Firstly, I would like to sincerely thank Professor Tadao Nakamura, my supervisor, of Tohoku University. I can never express enough gratitude for his guidance and encouragement to accomplish this thesis. He has also given me valuable advice in academic life and precious opportunity to work at Tohoku University.

I also wish to express my great appreciation to Professor Yoshiaki Nemoto and Professor Hiroki Shizuya of Tohoku University for their careful review and helpful suggestions.

I would also like to express my sincerely gratitude towards Associate Professor Hiroaki Kobayashi for his precise comments and enthusiastic discussions. Without his advice in this study and sensible insight into sciences, none of this research would be achieved.

Special thanks to Associate Professor Masayuki Katahira of Akita University and Assistant Professor Hong Shen of Tohoku Bunka Gakuen University for their kindly guidance to accomplish this research.

I wish to deeply thank Dr. Nobuyuki Oba of IBM Japan Ltd., for his encouragement. I gratefully acknowledge helpful discussions with Assistant Professor Kenichi Suzuki of Miyagi National College of Technology and Research Associate Hitoshi Yamauchi of The University of Electro-Communications on several points in this research.

I also acknowledge intellectual exchanges with Research Associate Taira Nakajima of Tohoku University and Research Associate Hiroyuki Takizawa of Niigata University.

Thanks to Noriaki Mori, Kentaro Sano, Mitsuhiro Nakaizumi, Yoshinori Kimura, and Yoshiyuki Kaeriyama, who have afforded me their assistance to enhance and improve the quality of this study.

I am grateful to all members in Nakamura Laboratory for their interest and critical response on my research activities.

Finally, I must thank my parents and my fiancée for their warm encouragement and kindhearted support.

Hiroyuki Kitajima

Jan. 2000

1 Introduction

Of late, application programs become more complicated and data used for the programs also become larger. Since computer technologies have remarkably been developed to process large-scale programs with huge data, high-speed processing of the programs could easily be achieved. The requirement of the high-speed processing cannot, however, be satisfied because the complication of application programs and the development of computer technologies are synergistically enhanced. Besides, this synergism makes high-speed processing of programs more difficult to be achieved. It causes various problems in the aspect of practical uses, such as low program availability and low utilization of computer resources.

Parallel processing is one of the most efficient solutions to achieve high-speed processing of a program. However, It is not simple to accomplish parallel processing in general due to two intrinsic issues as follows[1]:

- division of a program and high-speed parallel processing of the divided programs
- management of data communication as a side effect of parallelizing a program

To solve these issues, it is needed to examine not only programs and parallel computer architectures but also the relationship between the two. H. J. Siegel emphatically repre-

sented this demand[2]:

> *A widespread misconception is that the two most important parts of the high-performance computing field are architecture and algorithm. However, the interface between them is a crucial issue as well.*

Therefore, for practical parallel processing of programs, the clarification of relationship between programs and parallel computer architectures is most indispensable.

1.1 Parallel Programming Models

There can be parallelism at various levels of a program for parallel processing; e.g., parallelism at the task level, the thread level and the instruction level[1][3]. Figure 1.1 illustrates the parallelism of a program. When a program is executed in a parallel processing fashion, tasks are generated from a program. After that, threads are generated from each task.

A task is comprehended as a component of a program. It is related to a memory-address region in memory-address space when executed on a parallel processing system. A memory-address region for a task is independent of the others. A *Task* is also called a *process*. At least we say so in here. Threads are dealt with as components of a task, and connected with a memory-address region that for a task. In general, a thread means a sequence of instructions to be processed in parallel with the other sequences. In parallel processing of a program, the parallelism at the thread level or the instruction level may be the lowest level parallelism of a program.

The parallelism at the thread and instruction levels may be more effective to parallel

processing of a program because of their primitiveness. Threads, each of which consists of a series of several instructions are generated from a task when a program is executed on a parallel processing system. This involves the problem that limitation of the parallel processing efficiency cannot completely be removed by the thread-level or instruction-level parallel processing of the program. In other words, if tasks are not sufficiently extracted from a program, it is impossible to make the parallel processing efficiency higher even with manipulating parallel threads. Accordingly, the improvement in parallel processing efficiency based on the task-level parallelism is more important for parallel processing of a program.

Regarding the task-level parallelism of a program, programming models are classified into two types: the SPMD (Single-Program Multiple-Data) programming model and the MPMD (Multiple-Program Multiple-Data) programming model[1][4]. In a program based on the SPMD programming model, component tasks are homogeneous. Homogeneity in the program means that the same code is executed by multiple tasks in the different data domains. On the other hand, in a program based on the MPMD programming model, component tasks are heterogeneous. Heterogeneity means that multiple tasks may execute different codes. The SPMD programming model is sometimes called *the data-parallel programming model*, and the MPMD programming model is also called *the functional-parallel programming model.*

The SPMD programming model more simply archives highly parallel processing of a program than the MPMD programming model dose. This is because parallelism of data is more easily extracted than parallelism of functions. Therefore, the SPMD programming model is more important and valuable for the parallel processing of practical application

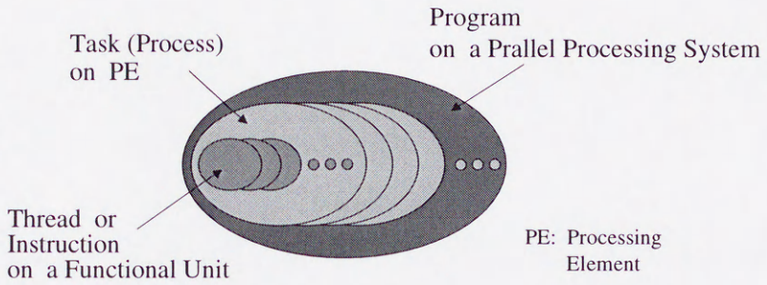


Figure 1.1: Parallelism of a program and a parallel computer architecture.

programs.

Besides, an SPMD program based on the SPMD programming model has comparatively high productivity than an MPMD program based on the MPMD programming model. Many application programs are based on the SPMD programming model because of the similarity with sequential programs in being written and understood. An MPMD program is not productive and convenient except the program is executed on a heterogeneous parallel-processing system. As a result, even though executed in functionally parallel, programs are designed based on the SPMD programming model in many cases. Programs specified in the MPMD programming model can also be rewritten into SPMD programs. The rewritten programs in the SPMD programming model become more understandable and convenient for users. We expect a number of application programs to be developed and improved under the SPMD programming model.

1.2 Parallel Computer Architectures

A parallel processing system consists of processing elements. A processing element includes functional units. Concerning parallel processing of a program on a parallel processing system, parallelism at each level in a program corresponds to parallelism of a component in the system; i.e., a processing element and a functional unit. Figure 1.1 also illustrates the correspondence of parallelism between in a program and in a parallel processing system. Parallelism of tasks generated from a program corresponds to parallelism of processing elements (PEs) in a parallel processing system[1][4]. Each processing element treats a task with a separated region in memory-address space. A *Processing element* (*PE*) is sometimes called a *computer* or *processor*. Parallelism among threads in a task corresponds to parallelism of functional units (processing units) in a processing element[5]. Each of functional units treats a thread. This is similar to the case of a processing element with a task except that units use the same memory-address region a processing element uses Parallelism at the thread level of a program can be used not only in a parallel processing system but in also a single PE (computer) system.

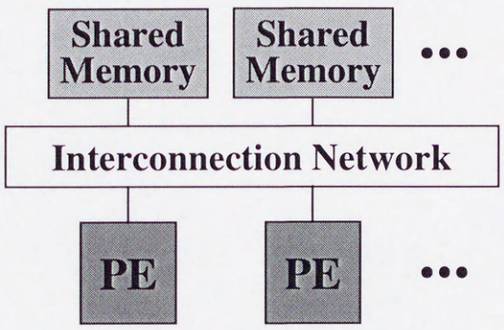
In respect of memory accesses from processing elements, parallel computer architectures for SPMD programs are roughly grouped into two kinds of architectures; i.e., shared-memory parallel computer architectures and distributed-memory parallel computer architectures[2][6]. Figure 1.2 illustrates these two parallel computer architectures. In the shared-memory parallel computer architecture, there is only one memory-address space. All processing elements can access the memory-address space, and pass data through the shared memory. In the distributed-memory parallel computer architecture, there are

multiple memory-address spaces. A processing element has its own local memory with a separated memory-address space[7]–[9], and can access only the local memory.

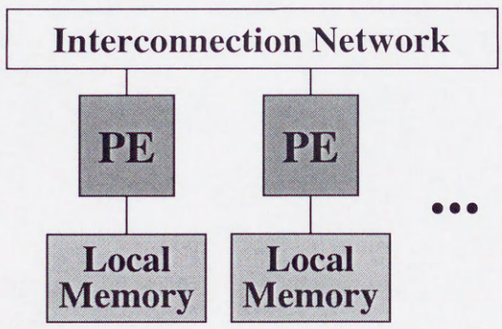
In the shared-memory parallel computer architecture, it is comparatively easy to maintain data coherence owing to its only one memory-address space. Parallel processing of a program is also simply accomplished in this architecture. Parallel processing efficiency of a program can, however, decrease due to access competition in a shared memory by processing elements. By contrast, in the distributed-memory parallel computer architecture, processing elements communicate data with synchronization mechanisms; e.g., a message-passing mechanism. Access competition does not occur in the distributed-memory parallel computer architecture due to individual memory-address spaces of processing elements. Nevertheless, it is impossible to avoid declining parallel processing efficiency of a program if processing elements frequently communicate with each other. This problem is caused by the communication overheads such as synchronization and establishment of the communication route among processing elements. Oppositely in the shared-memory parallel computer architecture, such problem can be solved with ordered use of a shared memory by processing elements. From these advantages, both the shared-memory parallel computer architecture and the distributed-memory parallel computer architecture must be examined for parallel processing of a program.

1.3 The Objectives of this Thesis

We focus on high-speed parallel processing of programs, especially, common application programs that are large-scale and complicated with a mass of data. As described in previous sections, analysis of SPMD programs in terms of practical use is important for



(a) The shared-memory parallel computer architecture.



(b) The distributed-memory parallel computer architecture.

Figure 1.2: Parallel computer architectures.

parallel processing of those application programs. Examination of the parallel computer architectures (i.e., the shared-memory parallel computer architecture and the distributed-memory parallel computer architecture) is also essential for them. Therefore, the objective of this thesis is to clarify relationship between SPMD programs and parallel computer architectures. Solutions for high-speed parallel processing of the programs are pursued through this clarification.

Parallelism of tasks in an SPMD program is used as data-reference locality of tasks by processing elements in a parallel processing system. Sufficient use of the data-reference locality would be able to achieve high utilization of processing elements. It also keeps frequency of communications among processing elements low. As a result, high processing-efficiency of an SPMD program can be acquired. From this point of view, a classification of SPMD programs into three types of data reference is proposed. After that, parallel computer architectures suitable for the data-reference types of programs are discussed.

Parallel processing efficiency of programs on abstract parallel computer architectures has been studied by many researches[10]–[12]. Those evaluations in abstract environments are reliable for small and simple programs; i.e., benchmark programs. In practical environments, large-scale application programs are executed on parallel processing systems. There must be many differences in results of the evaluations between under the abstract environments and practical environments. On this account, parallel processing systems for SPMD programs are also introduced through analyzing data references of application programs.

Moreover, task scheduling strategies and algorithms to improve parallel processing effi-

ciency of SPMD programs are studied. Data references of programs are considered in these studies. This is for the reason that static strategies and conventional algorithms are not always effective to acquire high parallel-processing efficiency of large-scale SPMD programs. Several experiments on parallel processing efficiency of SPMD programs are carried out with proposed techniques. The experimental results clarify relationship between SPMD programs and parallel computer architectures.

1.4 Thesis Organization

This thesis is organized as follows:

Chapter 1 describes introduction of this thesis.

Chapter 2 provides a classification of programs based on the SPMD (Single-Program Multiple-Data) programming model. SPMD programs are classified into three types of data reference as follows: (1) the type of locality-changeable data references, (2) the type of completely-distributed/completely-related data references, and (3) the type of completely-distributed/partially-related data references. Besides, this chapter examines parallel computer architectures suitable for the data-reference types of SPMD programs.

Chapter 3 examines functional programs in FL as SPMD programs with locality-changeable data references, and proposes a hierarchical parallel reduction system for FL programs[13]–[16]. An FL program differs from traditional imperative programs with many appealing properties such as transparency of the semantics, simple verification, and high productivity. Inefficiency in its implementation on parallel processing systems has still prevented the program from wide acceptance. Data-reference locality of FL programs is

actively used on the proposed system. A dynamic task scheduling strategy for the program is also proposed after analyzing static task scheduling strategies[13][17]–[20]. We name the dynamic task scheduling strategy *the breadth-first/depth-first task scheduling strategy.* This dynamic strategy solves problems conventional naive static task scheduling strategies have. Then, the parallel processing system and task scheduling strategy proposed in this chapter are discussed through several experiments. Parallel computer architectures and parallel processing techniques are explained, which are appropriate for executing SPMD programs with locality-changeable data references.

Chapter 4 treats a data-parallel volume rendering program as an SPMD program with completely-distributed/completely-related data references[21][22]. Volume rendering is an efficient tool to analyze and understand volumetric data in many scientific applications such as medical imaging and computational fluid dynamics. Volume rendering is, however, time-consuming in general. An image construction system is described in this chapter, which is a distributed-memory parallel processing system for volume rendering[23][24]. One-to-all and all-to-one communications among processing elements occurs during executing the program. It is indispensable to reduce the communication time. The binary-swap method is applied to the data-parallel volume rendering[24]. Moreover, the adaptive volume subdivision is also proposed[23]. After that, some experiments in terms of parallel processing efficiency of the program are carried out. Through the experiments, we discuss parallel computer architectures and parallel processing methods suitable for SPMD programs with completely-distributed/completely-related data references.

In Chapter 5, a program of an active contour model is regarded as an SPMD program with completely-distributed/partially-related data references. An active contour

model is one of image processing techniques. This model uses a contour for extracting ROI (a region-of-interest) from various images such as natural images and medical images. In general, efficient parallel processing of the program is difficult to be achieved. Parallelism in the program and its suitable parallel computer architecture are examined at the beginning[25]. Next, an active contour model using local shape information of ROI is proposed[25][26]. Several experiments indicate parallel computer architectures and parallel processing techniques appropriate for SPMD programs with completely-distributed/completely-related data references.

Chapter 6 gives conclusions and future works.

2 The Data-Reference Types of SPMD Programs and Parallel Computer Architectures

2.1 Introduction

Parallelism of tasks in an SPMD program is generated from parallelism of data used by those tasks[1][4]. In respect of processing tasks on a parallel processing system, each processing element exploits locality of data references as data parallelism. Sufficient using data-reference locality would attain high utilization ratio of processing elements, and keep frequency of communications among processing elements low. High parallel-processing efficiency of a program can be acquired. Therefore, efficiently utilizing data-reference locality of tasks is essential in parallel processing of an SPMD program.

From this point of view, this chapter proposes a classification of SPMD programs into three types of data references at the beginning. Moreover, parallel computer architectures suitable for the data-reference types of SPMD programs are discussed.

2.2 The data-reference types of SPMD programs

A program generates tasks executed in parallel. Generating tasks constructs their dependencies. Utilizing regularity of task generation would simply acquire high parallel-

processing efficiency of a program. A program, however, has conditional branches, which implies alternative decisions (if-statements), loops and recursions. In addition, processing time of each task is not always even. These matters cause difficulty to statically find out the regularity. They also give rise to complexity of maintaining data coherence in parallel processing of a program, and yield high frequency of data communications among processing elements. As a result, high parallel-processing efficiency of a program is not simply acquired.

Impossibility of statically finding out data-reference locality of tasks means impossibility of estimating changes in the locality. There exist SPMD programs each of which data-reference locality changes during it is executed. These SPMD programs are classified into the first type; i.e., the type of locality-changeable data references. Figure 2.1(a) illustrates an example of their task dependency graphs. In Figure 2.1(a), a circle and a line represent a task and dependency between two tasks, respectively. Irregularity in task dependency is shown in this figure. This type of SPMD programs typically includes SPMD programs that ought to be executed in functional parallel and specified as MPMD programs.

To solve the problems of an SPMD program with locality-changeable data references, allocating not a task but a set of tasks to a processing element is effective. Task sets can be constructed by task dependency. Data-reference locality of tasks would sufficiently be utilized in processing elements by selecting appropriate size of task sets according to the number of parallel tasks and idle processing elements. Attaining high utilization ratio of processing elements and keeping frequency of communications among processing elements low can consequently acquire high parallel-processing efficiency of a program. Dynamic

change of task-set size according to the number of parallel tasks and idle processing elements is essential.

Programs in scientific field (e.g., programs of image processing, cryptography and phonetic processing) are representative SPMD programs as well. Data-references regularity in the programs can statically be estimated due to structural uniformity of their data used by tasks, and obtained as results by processing tasks. This would result in efficient parallel processing of these programs. However, in the parallel processing of SPMD programs with the data uniformity, multiple data passing simultaneously occurs. Data passing among tasks implies data passing among processing elements. This remarkably decreases parallel-processing efficiency of programs owing to increasing communications among processing elements.

These programs can be classified into two types of data-passing patterns. One type includes programs whose tasks pass data with each other in one-to-all and all-to-one patterns. The other has programs in which tasks can pass data in many-to-many pattern. Programs having the former patterns are called SPMD programs with completely-distributed/completely-related data references. This type of data reference is the second type in this thesis. The latter programs are also called SPMD programs with completely-distributed/partially-related data references in the third data-reference type.

Figures 2.1(b) and (c) illustrate examples of corresponding task dependency graphs about the SPMD programs. Figure 2.1(b) explains the one-to-all and all-to-one dependency among task. *One-to-all data passing* among tasks is often called *data scattering* from one task to all tasks. *All-to-one data passing* is similarly called *data gathering*. Tasks

that gather all data and/or scatter data to all tasks with properly partitioning are called scatter-gather task in this thesis. On the other hand, the partial dependencies among tasks are shown in Figure 2.1(c). *Many-to-many data passing* among tasks is sometimes called *parallel data exchanges* among tasks.

Not only accomplishing high-speed processing of parallel tasks but also making one-to-all and all-to-one communications as efficient as possible are required by SPMD programs with completely-distributed/completely-related data references. Reduction in communicated data and improvement in communication transaction processing would satisfy these requirements. By contrast, grouping tasks under their partial dependency would be valuable for SPMD programs with completely-distributed/partially-related data references. Passing data among the groups in parallel prevents parallel-processing efficiency of programs from declining.

Both in the second and third data-reference type of SPMD programs, multiple parallel-tasks are usually allocated to a processing element when the number of parallel tasks is larger than that of processing elements. Unfortunately, allocating multiple-tasks to processing elements causes irregularity in task-processing time. Moreover, concerning practical application programs, size of data used for executing a parallel task is not always even. This also brings about irregularity in task-processing time. Irregularity in task-processing time means load imbalance among processing elements. Load imbalance obviously decreases parallel-processing efficiency of programs. As a consequence, load balancing among processing elements with using size and structural uniformity of data is important for these two types of SPMD programs. The load balancing is a premise to achieve high parallel-processing efficiency of programs.

An SPMD program has ether irregularity or regularity in its task dependency that implies data-reference locality of tasks. Moreover, an SPMD program with regular task dependency has either completely-related or partially-related data references. Therefore, SPMD programs can be classified into the three types of data-reference locality. Programs with irregularity of task dependencies are classified into the first type of data reference; i.e., the locality-changeable type. Programs with completely-related data references are named SPMD programs with completely-distributed/completely-related data references in the second type. Programs with partially-related data references are called SPMD programs with completely-distributed/partially-related data references in the third type. Hence, three data-reference types of SPMD programs are summarized with features of the programs and techniques for their efficient parallel processing as follows:

- the locality-changeable type

  An SPMD program with locality-changeable data references has impossibility to estimate changes of data-reference locality during the program is executed. Dynamically changing size of task sets according to the number of parallel tasks and available processing elements would be efficient for the programs.

- the completely-distributed/completely-related type

  An SPMD program with completely-distributed/completely-related data references has one-to-all and all-to-one dependencies among tasks. Reduction of data communications among tasks would be indispensable. Improving transactions of data scattering and gathering would also be valuable.

- the completely-distributed/partially-related type

  An SPMD program with completely-distributed/partially-related data references can

pass data in many-to-many pattern among tasks. Grouping tasks under their partial dependency would be effective.

2.3 Classification of parallel computer architectures based on the data-reference types of SPMD programs

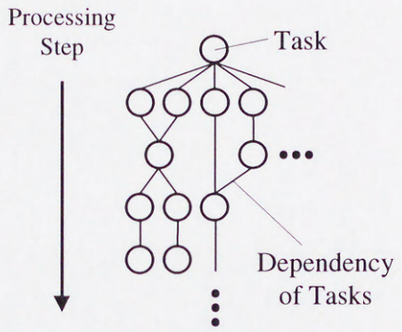
This section discusses parallel computer architectures suitable for the data-reference types of SPMD programs.

A parallel computer architecture for SPMD programs with locality-changeable data references

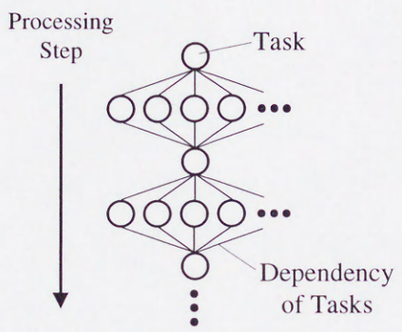
Estimating changes in data-reference locality of tasks is difficult during an SPMD program with locality-changeable data references is executed. This leads to complexity of maintaining data coherence and keeping frequency of data communication among processing elements low. Unique memory-address space supplied by a shared memory and data passing through the memory would relieve these problems.

In addition, dynamic changing task-set size would be efficient for this type of SPMD programs as described in previous section. With respect to the distributed-memory parallel computer architecture, processing elements can independently treat proper-sized task sets in their separated memory-address spaces. Each processing element makes good use of data-reference locality of task sets.

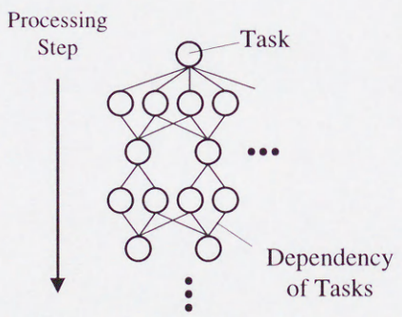
Therefore, the distributed-memory parallel computer architecture with a shared memory would be suitable for an SPMD program with locality-changeable data references.



(a) An SPMD program with locality-changeable data references.



(b) An SPMD program with completely-distributed/completely-related data references.



(c) An SPMD program with completely-distributed/partially-related data references.

Figure 2.1: Examples of task dependency graphs.

Besides, a hierarchical memory structure would also be valuable; e.g., cluster caches and memories between local memories and a shared memory. Data-reference locality is utilized at more various levels of a hierarchical memory structure.

A parallel computer architecture for SPMD programs with completely-distributed / completely-related data references

Achieving the high-speed processing of parallel tasks is principal for an SPMD program with completely-distributed/completely-related data references. Data scattering and gathering among tasks are also made as efficient as possible. The distributed-memory parallel computer architecture is proper for this type of programs with respect to high-speed processing of parallel tasks. Independent environments for executing tasks are supplied by local memories.

In addition, an interconnection network through which processing elements can locally communicate with each other would be valuable against the problem of data scattering and gathering. If multiple communications among processing elements are processed in parallel, data scattering and gathering are efficiently carried out with multistage communications such as binary-tree communication.

As a result, the distributed-memory parallel computer architecture with an interconnection network would be suitable for SPMD programs with completely-distributed/ completely-related data references.

A parallel computer architecture for SPMD programs with completely-distributed / partially-related data references

The completely-distributed/partially-related type of SPMD programs is similar to the completely-distributed/completely-related type. Perfect-parallel processing of tasks is accomplished when the data reference is completely distributed. Moreover, a number of local data passing among processing elements can be treated in this type of SPMD programs. Therefore, the distributed-memory parallel computer architecture with an interconnection network is appropriate to the SPMD programs. Multiple communications among processing elements should be carried out in parallel through an interconnection network.

Assume multiple tasks are allocated to a processing element. If tasks passing data with each other are allocated in a processing element, any communications among processing elements do not occur. Parallel-processing efficiency of a program does not decrease. It must be noted that allocating multiple tasks gives rise to a tradeoff between efficiency with less communications and degree of parallelism.

On the other hand, concern if all tasks that pass data with each other cannot be allocated to the same processing element. Part of data obtained by executing tasks in one processing element is passed to other processing elements. After preferentially executing tasks whose results are to be passed, each processing element may communicate the results and execute remaining tasks simultaneously.

The simultaneous treatment of communication and task execution may be included into improvements in data scattering and gathering for SPMD programs with completely-distributed/completely-related data references. For instance, parallel bi-directional com-

Table 2.1: The data-reference types of SPMD programs and their suitable parallel computer architectures.

Data-Reference Types of SPMD Programs	Appropriate Parallel Computer Architecture
locality-changeable	distributed-memory with a shared memory
completely-distributed / completely-related	distributed-memory with an interconnection network
completely-distributed / partially-related	distributed-memory with an interconnection network (simultaneous processing of task and communication)

munications is also effective for both SPMD programs with completely-distributed/completely-related and completely-distributed/partially-related data references. These improvements with an interconnection network offer uniform communications among processing elements due to task homogeneity of SPMD programs.

Table 2.1 summarizes the data-reference types of SPMD programs and their corresponding parallel computer architectures.

Related work

The relationship between programming models or structures of programming languages and abstract parallel-computer architectures has been studied by some researches[10]–[31]. Concerning parallel processing of a program on an abstract parallel-computing system, the system performance and parallel-processing efficiency of the program are evaluated under many assumptions and approximations regarding the parameters of parallel computer architectures they have. The parameters are, for instance, the number of processing elements, frequency of a system clock, calculation power of a processing element, load balance among processing elements, time to prepare a communication, communication facility among processing elements.

The evaluations of programs and architectures in abstract environments are reliable in small and simple programs such as "toy" programs. Nevertheless, with respect to practical execution of a large application program on a parallel processing system, there always are many differences between the evaluation results in the abstract environments and those in practical environments. This problem arises from difficulties of parameterizing various factors in the practical environments (e.g., load imbalance and irregularity in data communications among processing elements) due to changes in data-reference locality. Hence, the classification of SPMD programs based on the data-reference types, and the clarification of relationship between SPMD programs and parallel computer architectures discussed in this thesis would be more essential to estimate parallel processing efficiency of large and practical application programs.

Besides, the task-level parallel processing of a program is usually studied as tasks

scheduling strategies[32]–[43]. These strategies are heuristic in general because optimization problem of task scheduling is non-polynomial (NP) complete[44][45]. Many of the researches simplify particular scientific programs and benchmark programs, and limit target architectures or use their specialized architectures. In some of the researches on practical programs, the results of tracing programs and simplified parallel-computer architectures are utilized[46]–[49]. The proposals in this thesis (i.e., the classification of SPMD programs based on the data-reference types, and the clarification of relationship between SPMD programs and parallel computer architectures) have higher availability in the meaning of extensively supplying the classification and relationship to these researches.

2.4 Conclusions

This chapter has proposed a classification of SPMD programs into three types of data references at the beginning. The data-reference types represents as follows: 1) the type of locality-changeable data references, 2)the type of completely-distributed/completely-related data references, and 3) the type of completely-distributed/partially-related data references. After that, parallel computer architectures suitable for the individual data-reference types have been discussed. Consequently, the relationship between the data-reference types of SPMD programs and parallel computer architectures are obtained as follows:

1) the locality-changeable type and
the distributed-memory parallel computer architecture with a shared memory

2) the completely-distributed/completely-related type and

the distributed-memory parallel computer architecture with an interconnection network

3) the completely-distributed/partially-related type and
the distributed-memory parallel computer architecture with an interconnection network (allowed simultaneous processing of task execution and communication in processing elements)

In the following chapters, the data-reference types of SPMD programs are investigated with practical application programs. Besides, parallel processing systems for the individual application programs are proposed. Some techniques to improve parallel processing efficiency of these SPMD programs are also examined.

3 A Parallel Computer Architecture for SPMD Programs with Locality-Changeable Data References

3.1 Introduction

Regarding an SPMD program with locality-changeable data references described in Chapter 2, this chapter discusses a distributed-memory parallel computer architecture with a shared memory and task scheduling strategies to change the task-set size dynamically during the execution of the program.

In this chapter, a functional program is examined as the data-reference type of an SPMD program. Functional programs differ from traditional imperative ones with many appealing properties such as transparency of the semantics, simple verification and high productivity[50]. The inefficiency in their implementation on parallel processing systems has still prevented the program from wide acceptance.

At the beginning, this chapter briefly explains functional programs, and proposes a hierarchical parallel processing system for the program, which based on the distributed-memory parallel computer architecture with a shared memory[13]–[16]. Through analyzing static task scheduling strategies, a dynamic task scheduling strategy to make good use of the data-reference locality is also proposed in this chapter[13][17]–[20]. Moreover, the per-

formance of the parallel processing system with the task scheduling strategies is discussed through several experiments.

3.2 An SPMD Program with locality-changeable data references and its parallel processing system

3.2.1 Functional language FL

Functional programs (e.g., Haskell[51], Miranda[52], FL[53]–[55], and LML[56]–[58]) differ from traditional imperative ones with many appealing properties such as transparency of the semantics, simple verification and high productivity. FL has been proposed as a practical functional language based on FP[50][59]. FL not only inherits the basic characteristics and advantages of FP but also offers programmers more convenience while keeping clear semantics.

An FL program consists of primitive functions, data used by the functions, and combining forms. Primitive functions include arithmetic operations, list operations, predicates, and so on. Combining forms signifies higher-order functions whose arguments are primitive functions. Figure 3.1 shows a typical example of FL programs, which performs computation of matrix multiplication. The data should have the format of $\langle \mathbf{A}, \mathbf{B} \rangle$, where $\mathbf{A}$ and $\mathbf{B}$ are matrices with the size of $i \times k$ and $k \times j$, respectively. Appendix A will give a part of FL primitive functions, and combination forms to explain the example.

From the programmer's view, an FL program is a function that usually consists of a set of functions and is applied to data to obtain new data. A pair of a primitive function and data used by the function can be a task. In addition, the dependency of tasks must

uniquely be determined according to the structure of functions and data used by the functions. This clearly explains that the data-reference locality is dynamically changed during the execution of the program. Hence, a functional program is an SPMD program with locality-changeable data references.

The dependency among tasks in an FL program is often represented with a reduction graph[60]. *Reduction* means the execution of a task in an FL program. Figure 3.2 illustrates the reduction graph of the matrix multiplication program shown in Figure 3.1. Of course, the reduction graph proves the determinacy of task dependency, which implies the data-reference locality and the potential of parallel task processing, in an FL program.

```
def mmul ≡ α:(α:IP) ∘ (α:distl) ∘ distr ∘ [s1, trans ∘ s2] where
{
    def IP ≡ add ∘ (α:mul) ∘ trans
}.
```

Figure 3.1: Example of an FL program (matrix multiplication).

3.2.2 An FL hierarchical parallel reduction system

In the parallel processing of an SPMD program with locality-changeable data references, it is difficult to estimate the changes in data-reference locality during the execution. This causes the complexity to maintain the data coherence in parallel processing of the program, and keep the frequency of data communication among processing elements low. As described in Section 2.3, the distributed-memory parallel computer architecture with a

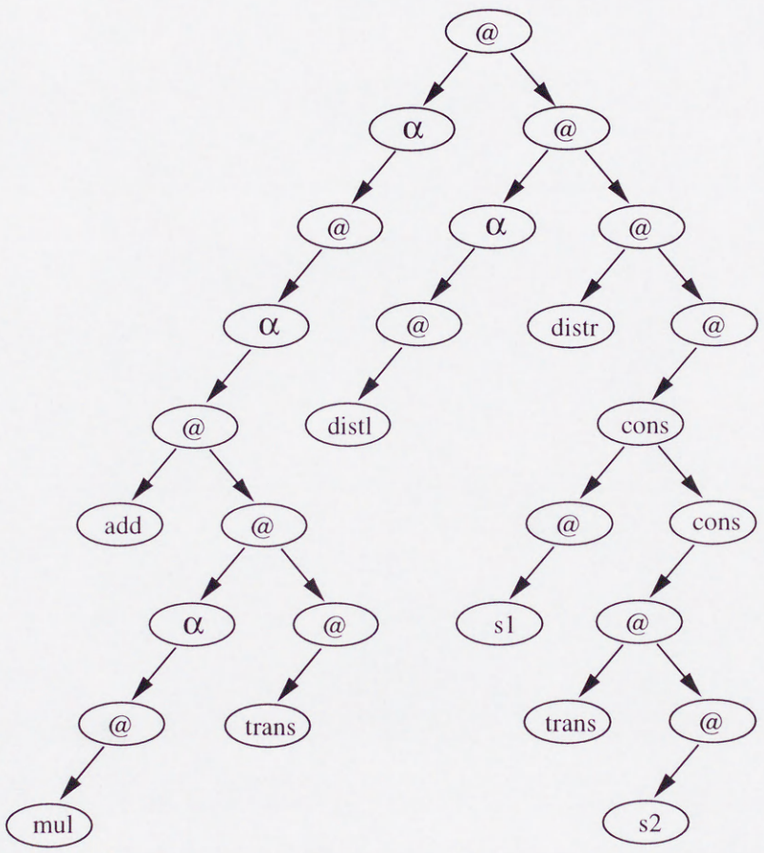


Figure 3.2: Graph expression of an FL program (matrix multiplication).

shared memory would be suitable for this type of SPMD programs. In this architecture, it can also be effective to utilize caches between local memories and a shared memory such as cluster caches because the data-reference locality can be utilized at various levels of a hierarchical memory structure. Figure 3.3 illustrates a parallel computer architecture appropriate for the SPMD programs.

To confirm the relationship between the SPMD program and parallel computer architecture, this chapter proposes a hierarchical parallel reduction system for an FL program, which is based on the distributed-memory parallel computer architecture with a shared memory. Figure 3.4 gives a block diagram of the system. This system is constructed for the sake of maintaining the data coherence implicitly and utilizing the data-reference locality actively. Moreover, the processing elements in this system are coupled through cluster caches to use the data-reference locality sufficiently.

In order to examine the data-reference locality, this system treats a task separately as a pair of a primitive function and its data. The shared-memory modules are divided into two groups according to the components of a task, resulting in the shared-memory module having two address spaces. One group, called a *Graph Memory*, is used for storing program codes including primitive functions and combination forms. The other, called a *Data Memory*, is for storing data. The treatment of a task is similar to common processing systems, which treat threads or instructions, and data separately.

In addition, the delay in allocating primitive functions to processing elements should be reduced as much as possible. Parallel processing of a FL program conceptually consists of many rewriting operations that repeatedly rewrite the component functions in different

parts of a program with data in parallel. Besides, the overall transactions of processing a task can be divided into three stages; i.e., task detection, task allocation and task execution. The transactions of a task are applicable to a primitive function. Therefore, the transactions in these three stages can be overlapped in terms of time as a high-level processing pipeline in the system. Primitive functions that can be executed simultaneously in a program are detected by a *Scheduler* in the first stage of the pipeline. The detection of functions is held according to a parallel execution strategy named FGK'[61]. The detected functions are scheduled into queues in a *Task Pool* in the second stage of the pipeline. Each queue is corresponding to its processing element. Finally, the allocated functions are simultaneously executed in multiple processing elements of the system.

By contrast, data is supplied to each processing element from the hierarchical memories; i.e., its local memory, a cluster cache and a *Data Memory* as a shared memory. The hierarchical memory structure would enable processing elements to utilize the data-reference locality of tasks efficiently while the shared memory is implicitly maintaining data coherence.

So as to simplify the pipeline synchronization mechanism, the time of the slowest stage in the pipeline is chosen as the cycle time of the pipeline. In the meantime, tasks are executed in each processing element asynchronously. For instance, the execution of some primitive functions like "trans" may need two or more pipeline cycles. Activities between tasks are coordinated under a pipeline runtime support system through the *Scheduler*. Figure 3.5 depicts the execution diagram when mapping the matrix multiplication program to the hierarchical parallel reduction system with the data size $i = 2$, $j = 3$ and $k = 1$. The three axes in Figure 3.5 correspond to time, reduction pipeline stage and the number

of processing elements, respectively. Tasks processed in each pipeline cycle are listed out, and the cycles with mark $\times$ mean that the system runs idle because of data-dependency. It should be noted that, in fact, the processing time of tasks is not even and it takes various access delays from processing elements to the memories.

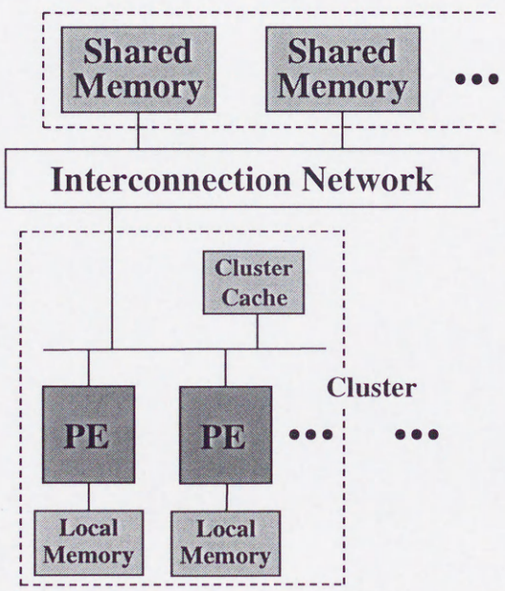


Figure 3.3: Parallel computer architecture for FL programs.

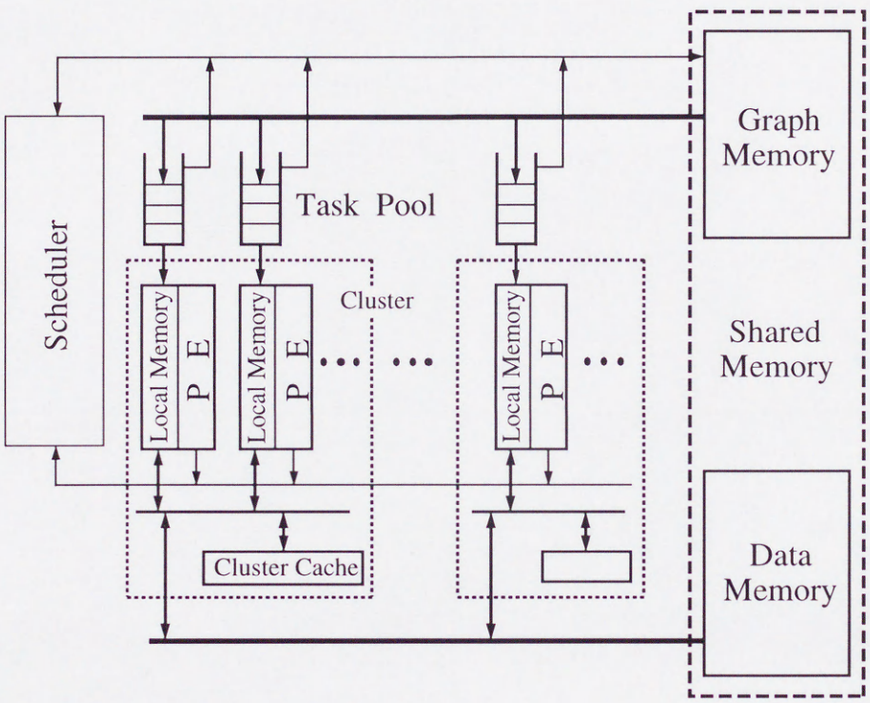


Figure 3.4: Block diagram of the hierarchical parallel reduction system.

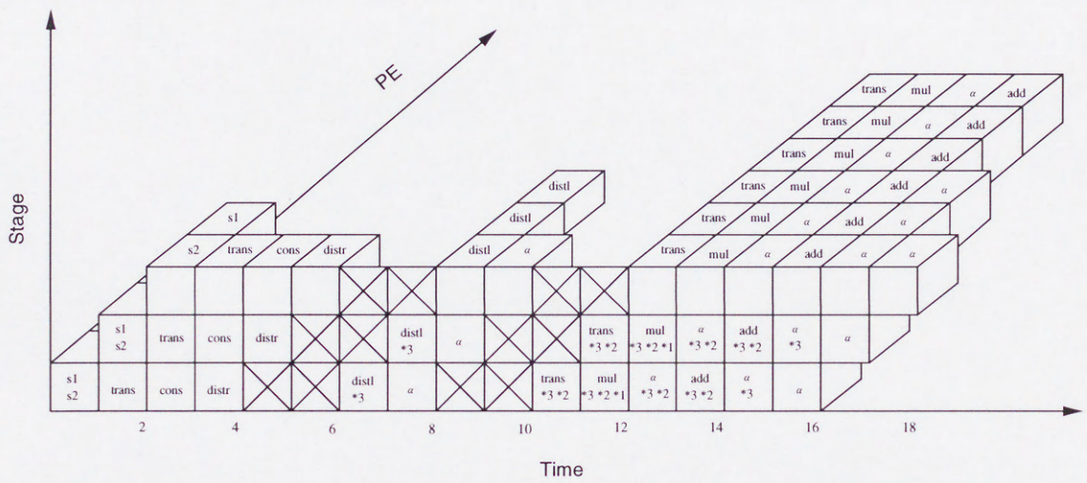


Figure 3.5: Hierarchical parallel reduction of an FL program (matrix multiplication).

3.3 Parallel processing of an SPMD program with locality-changeable data references on its parallel processing system

3.3.1 A breadth-first / depth-first task scheduling strategy with locality consideration of data references

The dynamic change in task-set size discussed in Section2.2 would be needed for an SPMD program with locality-changeable data references since it enabled each processing element to treat a proper-sized task set locally. The purpose of task scheduling in a parallel processing system is to maintain load balancing with the consideration of the data-reference locality and obtain high utilization of processing elements. Based on analysis of some static task scheduling strategies, this section proposes a dynamic task scheduling strategy for FL programs as SPMD programs with locality-changeable data references.

Static task scheduling

Representative static task scheduling strategies are the breadth-first task scheduling strategy and the depth-first task scheduling strategy. Figure 3.6 shows an allocation example when the breadth-first task scheduling strategy is applied to a generalized task-dependency graph on a system with 8 processing elements. In the figure, a/b of each graph node means that processing element a executes the node in pipeline cycle b.

In addition, the right-hand node of each node is given higher priority to be searched and detected tasks are allocated to as the same processing elements as possible in which the predecessor tasks are executed. It can be found that this scheduling strategy extracts parallelism as much as possible so that the processing elements can be sufficiently uti-

lized. It is, however, impossible for the strategy to use data-reference locality of tasks effectively because the parallel tasks have the priority to be detected over the sequential tasks. Figure 3.6 denotes that the data-reference locality cannot be utilized in the first and second pipeline cycles while parallel tasks are detected as much as possible. These results in the high frequency of communication among processing elements for data exchanges. Consequently, high parallel-processing efficiency of a program could not be obtained.

On the other hand, the depth-first task scheduling strategy can reduce the frequency of data communication among processing elements as much as possible due to sufficient use of the data-reference locality. Figure 3.7 illustrates an allocation example when the depth-first task scheduling strategy is applied to a generalized task-dependency graph on a system with 8 processing elements. For instance, the processing elements 1, 3, 5 and 7 execute the sequential tasks in this figure, while no processing elements execute the sequential tasks in Figure 3.6. Conversely, it is difficult to achieve high-speed processing of a program because the depth-first task scheduling strategy does not detect parallel tasks enough. This is caused by that the sequential tasks have the priority to be detected over the parallel tasks. In Figure 3.7, processing time of the task-dependency graph (i.e., 7 pipeline cycles) is longer than the one (i.e., 6 pipeline cycles) in the case of the breadth-first task scheduling strategy.

Dynamic task scheduling

Although the breadth-first task scheduling strategy extracts parallelism as much as possible, it is impossible for this strategy to use data-reference locality of tasks effectively.

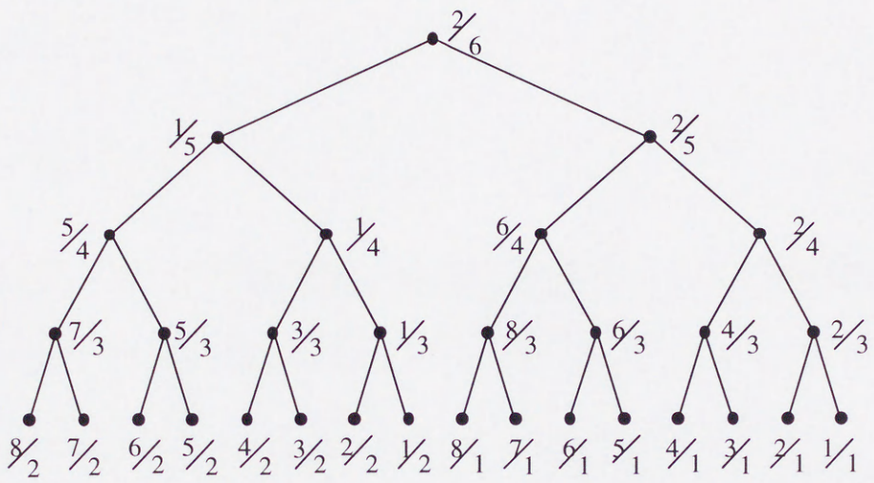


Figure 3.6: Breadth-first task scheduling strategy.

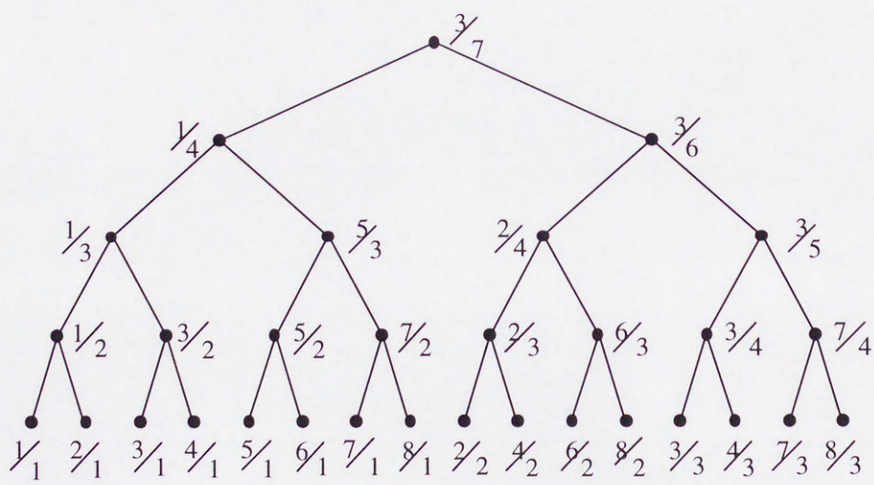


Figure 3.7: Depth-first task scheduling strategy.

The depth-first task scheduling strategy can sufficiently use the data-reference locality, but does not detect parallel tasks enough. By contrast with these static task scheduling strategies, the breadth-first/depth-first task scheduling strategy is proposed, which is a dynamic task scheduling strategy. The breadth-first/depth-first task scheduling strategy makes use of the breadth-first mode for parallel task searching. The breadth-first/depth-first scheduling strategy also develops the depth-first scheduling mode as a complement to the breadth-first mode. When the number of branches searched exceeds a certain threshold that is set according to the number of idle processing elements, the reduction graph will be traversed in a depth-first manner.

Figure 3.8 gives an example when the traverse of a generalized task-dependency graph is switched from the breadth-first mode to the depth-first one on the system with 8 processing elements, where the default threshold is 8. As shown in Figure 3.8, after the mode switching when the number of the searched tasks exceeds the threshold, tasks are generated by traversing the graph depth-first. Each processing element can execute tree tasks that have the local data references in pipeline cycle 1, 2 and 3. As a result, not only are the processing elements taken advantage of, but also the data-reference locality is taken care of. This means that the breadth-first/depth-first task scheduling strategy can accomplish the dynamic change of the data-reference locality and the size of task sets. The proposed strategy is expected to have the higher efficiency for an FL program.

In the hierarchical parallel reduction system proposed in this chapter, tasks to be allocated to the same processing element are always connected as the dependency between tasks. According to the dependency, the scheduler can determine whether to allocate the successor task to the same processing element or not. It should be mentioned that the

scheduler takes the data-reference locality into account as a secondary factor for task allocation. The first one is the availability of each processing element to accept tasks. This priority order is decided for achieving high-speed processing of a program.

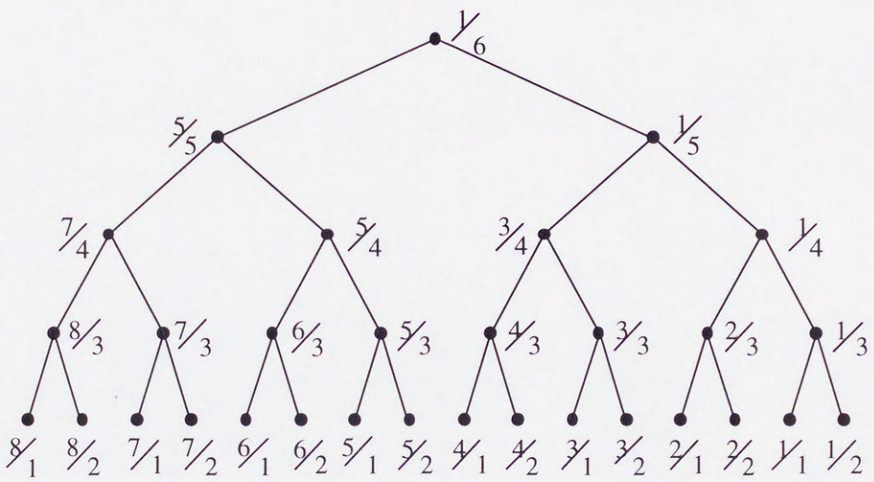


Figure 3.8: Breadth-first/Depth-first task scheduling strategy.

Related work

Although some scheduling strategies previously reported for functional architectures[62]–[65] have gained inspiring achievements with load balancing and good system utilization, they have not sufficiently drawn the parallel processing efficiency of programs because it is still difficult to maintain the locality of memory references. Besides, other researchers have reported dynamic task scheduling strategies; i.e., the LIFO task scheduling strategy[66] and the LIFO/FIFO task scheduling strategy[67].

Since LIFO is based on the breadth-first strategy, the utilization ratio of processing elements is limited due to long waits of tasks detected from a program. LIFO/FIFO makes the size of a task set as large as possible. Hence, in the LIFO/FIFO task scheduling strategy, if parallel tasks are frequently generated during the execution of a program, the overhead of task allocation to processing elements is larger than the one of the breadth-first/depth-first task scheduling strategy proposed in this thesis, and the task parallelism is not sufficiently used.

3.3.2 Performance evaluation

This section carries out simulation experiments of the hierarchical parallel reduction system to discuss a parallel computer architecture and parallel processing techniques for SPMD programs with locality-changeable data references.

Parameters for experiments

The simulation experiments are based on the architecture shown in Figure 3.4. The compiled FL programs are stored in the *Graph Memory*, and the selected primitive functions are temporarily stored into the queues of the *Task Pool* as executable tasks. The *Scheduler* is monitoring the states of the queues and the processing elements. It controls the detection and the allocation of primitive functions from the memory to the processing elements via the corresponding queues. Processing elements perform the execution of the allocated functions. Some execution operations may take several pipeline cycles. The blocks of data in the *Data Memory* are transferred to/from the cluster caches with the direct mapping

policy.

A number of various FL benchmark programs are executed in the experiments. Table 3.1 describes the benchmark programs. These programs are selected from a wide range of applications that can be divided into two classes. The one is n-queen problems: A∼E. The other is matrix multiplication programs: F∼J. Notice that the number of tasks generated in these programs varies from hundreds to thousands and their average parallelism varies from ten to twelve thousands. Here, the average parallelism of each program indicates the potential parallelism of a program, and shows the maximum speedup that can be achieved on an unlimited number of processing elements[68]. Therefore, simulation results by using these benchmarks can be considered reliable.

The system performance under three task scheduling strategies is measured in terms of memory accesses and speedups of each program. The simulation results of the benchmark programs are measured on a clustered system with 4 clusters, each of which consists of 4 processing elements. Access latencies related to the memories and caches are given in Table 3.2. An access latency from each processing element to its local memory is 1 pipeline cycle. If data is not in a local memory and is in the cluster cache, it takes 2 more pipeline cycles. It sums up to 3 cycles for loading data to the processing element. The cluster cache miss penalty resulting in a *Data Memory* access is of 6 cycles.

Experimental results

Figure 3.9 shows the numbers of accesses in programs A and F under three task scheduling strategies. In each program, it can be found that the breadth-first/depth-first task

Table 3.1: Benchmark programs.

	Programs	Number of Tasks	Average Parallelism
A	4-queens	4099	17.369
B	5-queens	19793	65.757
C	6-queens	101693	277.850
D	7-queens	482697	1119.947
E	8-queens	2473187	4986.264
F	10 × 10	2562	122.000
G	20 × 20	18112	862.476
H	30 × 30	58662	2793.429
I	40 × 40	136212	6486.286
J	50 × 50	262762	12512.476

Table 3.2: Access penalties to memories and a cache.

Case	Data in Local Memory	Data not in Local Memory	
		Cluster Cache hit	Cluster Cache miss (Data in Data Memory)
Latency in Cycles	1	3	7

scheduling strategy has obviously increased the number of local memory accesses compared with the other strategies. The experimental results clearly show that the data-reference locality of programs is fully utilized through local memories. The more sufficient use of the local memory means the more appropriate selection of the task-set size. These results also imply the efficient use of both local memories to utilize the data-reference locality and the *Data Memory* to maintain the data coherence without extra communication. The access ratio of local memories also varies approximately from 95% to 60% in the other benchmark programs. Even in these cases, similar efficiency of the system and the proposed strategy are obtained.

In Figure 3.10, their efficiency is also depicted as speedup ratios of a program to the performance of the 1 processing-elements system. It is found that the more the number of local memory is, the larger the speedup is. Especially with large programs, the speedups are close to the parallel processing ability of the system.

In conclusion, the hierarchical parallel reduction system and the breadth-first/ depth-first task scheduling strategy are remarkably valuable to obtain the higher parallel-processing efficiency of FL programs. Therefore, as concerns SPMD programs with locality-changeable data references, the experimental results reveal the effectiveness of the distributed-memory parallel computer architecture with a shared memory and the indispensability of a task scheduling strategy to change the task-set size dynamically during the execution of a program.

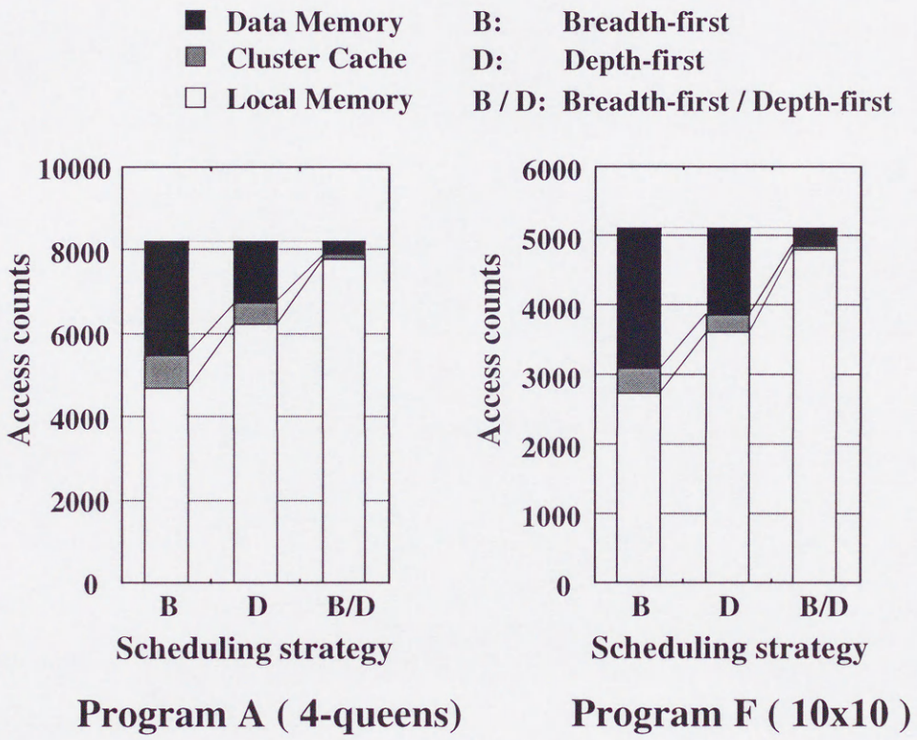


Figure 3.9: Number of accesses to the memories and caches.

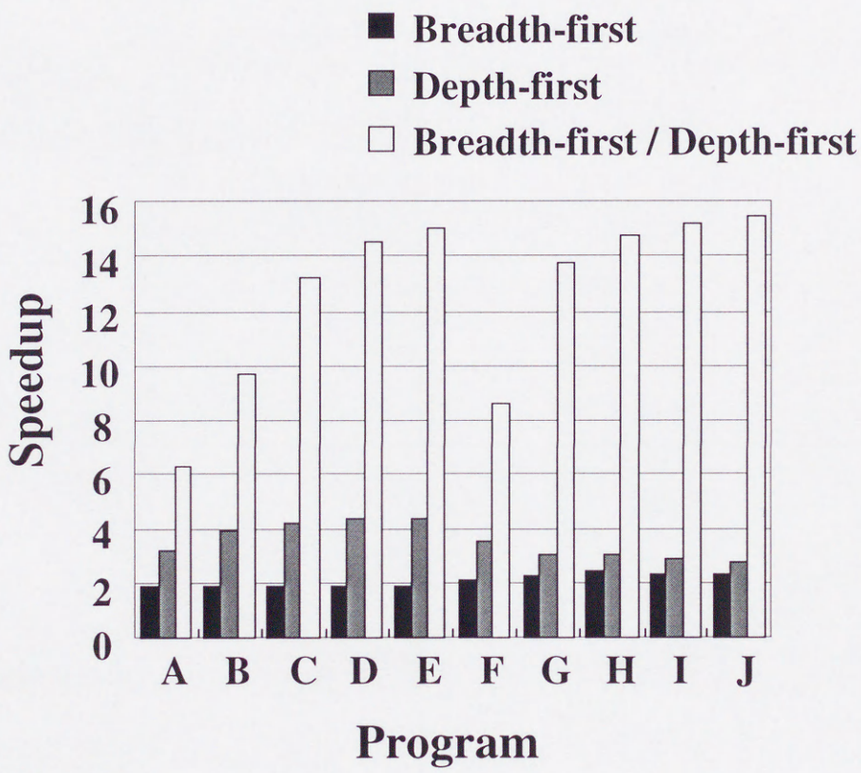


Figure 3.10: Speedup in benchmark programs.

3.4 Conclusions

This chapter has treated functional programs as SPMD programs with locality-changeable data references, and proposed a hierarchical parallel reduction system for this kind of programs. This system is based on a distributed-memory parallel processing system with a shared memory for active use of data-reference locality. Through the analysis of task scheduling strategies for general programs, this chapter has also proposed a dynamic task scheduling strategy for the execution of FL programs on the proposed parallel reduction system, which is called the breadth-first/depth-first task scheduling strategy.

The effectiveness of the parallel processing system and the dynamic task scheduling strategy has been examined through several experiments. The experimental results show that the breadth-first/depth-first task scheduling strategy proposed in this thesis has a capability of both parallel processing oriented scheduling and locality-considered scheduling according to the utilization ratio of processing elements and the parallelism of tasks generated from the program.

Therefore, as regards SPMD programs with locality-changeable data references, the distributed-memory parallel computer architecture with a shared memory is appropriate and task scheduling strategies to change the size of task sets dynamically is essential.

4 A Parallel Computer Architecture for SPMD Programs with Completely-Distributed / Completely-Related Data References

4.1 Introduction

Concerning SPMD programs with completely-distributed/completely-related data references described in Chapter 2, this chapter discusses the distributed-memory parallel computer architecture with an interconnection network and techniques to reduce the amount of communications among processing elements and also to improve the transactions for data scattering and gathering.

This chapter treats data-parallel volume rendering as a class of SPMD programs with completely-distributed/completely-related data references. Volume rendering is an efficient tool to analyze and understand volumetric data in many scientific applications such as medical imaging and computational fluid dynamics[69]. Volume rendering is, however, time-consuming in general and many researchers concentrate on accelerating and parallelizing it.

This chapter begins by describing data-parallel volume rendering and a distributed-memory parallel processing system with an interconnection network for volume rendering[21]–[24]. Next, with improving the communication transactions for data scattering and gath-

ering, this chapter proposes an adaptive volume subdivision method to reduce data passed among processing elements and still maintain the load balance among them[23][24]. The effectiveness of the parallel processing system and data-parallel volume rendering with the adaptive volume subdivision method are examined by some experiments.

4.2 An SPMD program with completely-distributed / completely-related data references and its parallel processing system

4.2.1 Data-parallel volume rendering

Volume rendering[69] is an efficient tool for analyzing and understanding volumetric data in many scientific applications such as medical imaging, molecular structuring and computational fluid dynamics. Many researchers have concentrated on real-time rendering at subsecond image-update rates to explore their data. However, the use of volume rendering has been restricted because it is still computationally expensive.

A number of algorithms have been proposed for acceleration of volume rendering[70]–[72] on a single processing-element system. For instance, Figure 4.1 illustrates representative volume rendering with the ray-casting algorithm[70]. In ray-casting volume rendering, a viewing-ray is cast through each pixel on a screen from a viewing point. The pixel value is calculated with the rendering equation to sample points along the viewing-ray. Each sampling point has an intensity and an opacity calculated from the neighboring pre-defined voxels.

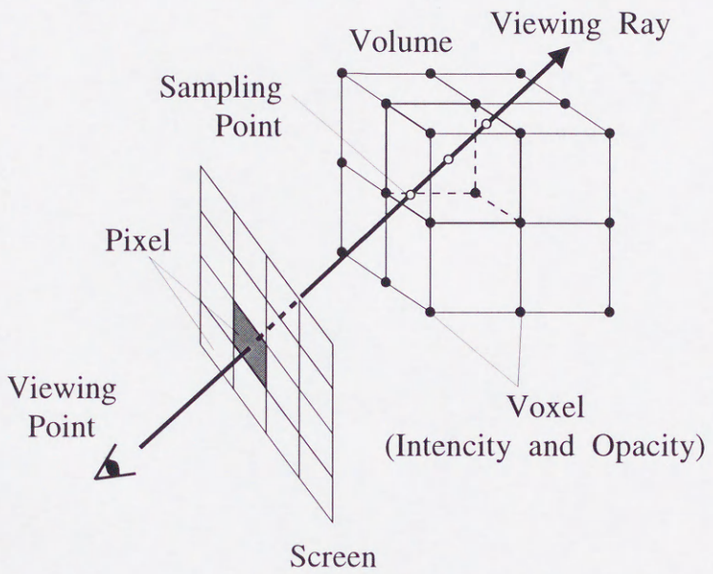


Figure 4.1: Ray-casting volume rendering.

Voxel-parallelization of volume rendering

Many researchers proposes data-parallel processing techniques for the acceleration of volume rendering[73]–[84]. The data-parallelizations of volume rendering are grouped into two classes. One is, parallelization based on pixels called *pixel parallelization* and the other is based on voxels called *voxel parallelization.*

A unit of pixel parallelization is usually a pixel on an image screen[77]–[80]. Volume rendering for a pixel value is considered a task allocated to a processing element. On the other hand, a unit of voxel parallelization is a voxel in a volume space. In voxel parallelization, a volume is divided into subvolumes, each of which corresponds to processing elements[82]–[84]. Rendering of a subvolume is a task. In both parallelizations,

all tasks are treated in processing elements simultaneously. In addition, all data of processing elements are gathered and/or scattered to obtain intermediate results and a final result. Hence, a program of data-parallel volume rendering is an SPMD program with completely-distributed/completely-related data references.

With respect to pixel parallelization, the independency in contributions of sampling points for pixels is exploited. For example, in a pixel-parallel volume rendering based on ray-casting volume rendering, the rendering equation along a viewing ray cast through a pixel is allocated into a processing element as a task. This task allocation into a processing element is carried out without difficulty. It is, however, not simple to estimate load balancing among processing elements with this parallelization because data to be used by tasks is not always uniform in a volume space. By contrast, in voxel parallelization, a volume can be divided into subvolumes with considering the amount of valuable data. Accordingly, it is easier to maintain the load balance among processing elements in voxel-parallel processing than in pixel-parallel processing. This implies that voxel parallelization can achieve higher parallel-processing efficiency of volume rendering with many processing elements. Therefore, voxel parallelization of volume rendering is focused in this thesis.

Parallel shear-warp volume rendering

Algorithms for accelerating volume rendering on single processing-element systems can broadly be classified into ray-casting[70], splatting[71], or shear-warp algorithms[72]. Among these algorithms, the shear-warp algorithms are the most promising because they can realize semi-interactive rendering of moderate-sized volumetric data on a conventional graph-

ics workstation without affecting image quality. Nevertheless, even though a shear-warp factorization algorithm is used, volume rendering of a large volume data set is still time consuming and it is difficult to realize interactive rendering rates on a single processing-element system.

Consider the voxel parallelizing of the shear-warp factorization algorithms. Hereafter, this research distinguishes parallelized shear-warp factorization algorithms from the shear-warp factorization algorithms in [72], and calls the former the *parallel shear-warp algorithm* and the latter the *serial shear-warp algorithm.*

Serial shear-warp algorithm[72] is an object-order algorithm based on three stages as follows:

1. **Shear stage**

   shearing volume parallel to the slices of it with the viewing transformation matrix

2. **Projection stage**

   forming a distorted intermediate image by the projection of volume slices

3. **Warp stage**

   2D warp to form an undistorted final image

where the intensity and opacity of each voxel are calculated as preprocessing. Figure 4.2 depicts volume rendering by the serial shear-warp factorization algorithm. First, a volume is sheared by translating the slices so that viewing rays are perpendicular to the slices in the shear stage. The translation is performed by resampling each slice of the volume with the constant weight overall a slice. Next, translated slices are projected to the

intermediate image in the projection stage. Finally, the intermediate image with distortion is transformed into the correct final image. This transformation can be done by applying a 2D-affine warp. In this approach, projection similar to a ray-casting algorithm[70] is effectively achieved by projection in the sheared volume space and 2D transformation in the projected image space. A process of each stage is a large and time-consuming task. Task dependency of the serial shear-warp volume rendering is shown in Figure 4.3. In Figure 4.3, these three tasks are serially executed on a single processing-element system.

The parallel processing of shearing and projecting subvolumes is valuable for accelerating the serial shear-warp volume rendering. In addition, since a task that warps the distorted intermediate image may be a set of warping-tasks each of which has a piece of the intermediate image, parallel processing of the warp should be achieved to generate an undistorted final image for more accelerated volume rendering. This chapter calls the warping-task the *parallel-warp task*. In this case, there must be a scatter-gather task that gathers all results of parallel shearing and projecting, named subvolume images. The scatter-gather task also scatters the intermediate image to all processing elements after projecting the subvolume images. This chapter calls the scatter-gather task the *compositing task* and also calls this stage the *composition stage* of subvolume images to generate the intermediate image. The compositing task gathers and projects all subvolume images to generate the distorted intermediate image, and continuously scatters the intermediate image to all parallel-warp tasks after partitioning it. Moreover, another scatter-gather task that gathers all pieces of the undistorted final image and accumulates them is needed. This scatter-gather task and the stage are called the *merging task* and the *merge stage*, respectively. Figure 4.4 illustrates the parallel shear-warp volume rendering algorithm.

Since the volume partitioning to subvolumes can be included the preprocessing, the stages of parallel shear-warp volume rendering and the task dependency are summarized in Figure 4.5 and as follows:

1. **Parallel shear stage**

   Each task shears a subvolume and projects it to generate a subvolume image. Hereafter, this chapter calls the task *parallel shear task*. Each subvolume image only contains the partial contribution of a subvolume to an intermediate image.

2. **Composition stage**

   A compositing task gathers all subvolume images and carries out the projection of the subvolume images to generate a distorted intermediate image. After that, the partitioning of the intermediate image is carried out by the compositing task in this stage.

3. **Parallel warp stage**

   A piece of the distorted intermediate image is warped by the parallel-warp task to generate a piece of an undistorted final image.

4. **Merge stage**

   A merging task gathers all pieces of the undistorted final image and accumulates them to obtain a final image.

4.2.2 An image construction system for data-parallel volume rendering

In parallel processing of SPMD programs with completely-distributed/completely-related data references, it is principal to achieve the high-speed processing of parallel tasks, and

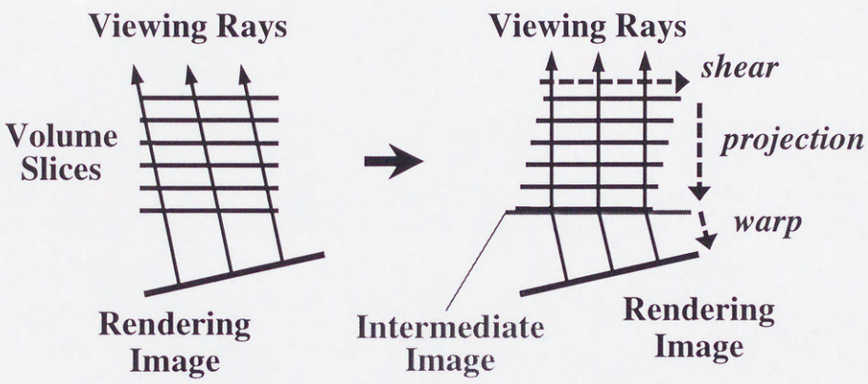


Figure 4.2: Serial shear-warp volume rendering.

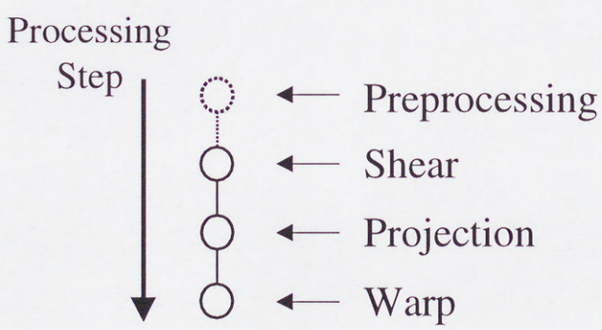


Figure 4.3: Task dependency in serial shear-warp volume rendering.

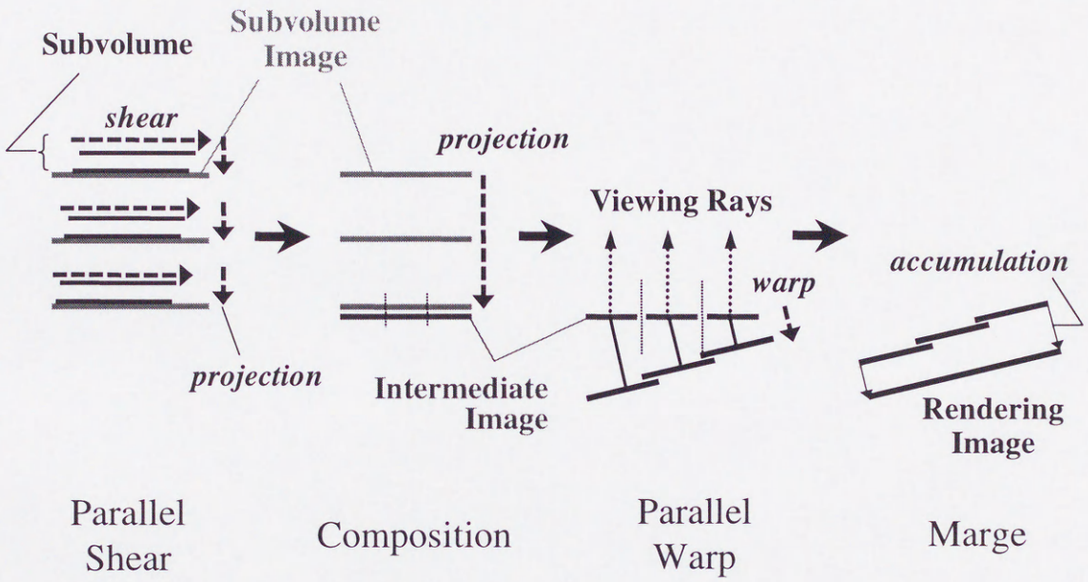


Figure 4.4: Parallel shear-warp volume rendering.

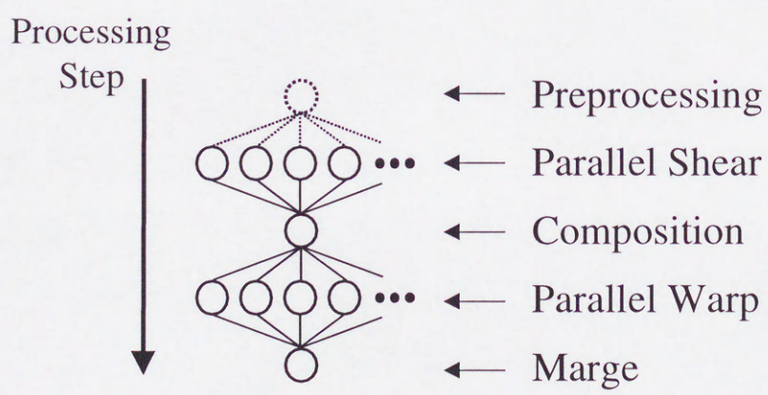


Figure 4.5: Task dependency in parallel shear-warp volume rendering.

to make the data scatter and gather as efficient as possible. As described in Section 2.3, the distributed-memory parallel computer architecture is proper for the SPMD programs to achieve the high-speed parallel processing. Against the data scattering and gathering, it would be valuable to employ an interconnection network that the processing elements can locally communicate with each other. If multiple communications among processing elements can be treated in parallel, the data scatter and gather are efficiently carried out with various communication schemes such as hypercube communications[85] and binary-tree communications. Therefore, the image construction system for data-parallel volume rendering this chapter concentrates is based on the distributed-memory parallel computer architecture with an interconnection network. Figure 4.6 illustrates the block diagram of the system.

If one scatter-gather task can be divided into some small scatter-gather tasks, multiple processing elements can execute the small scatter-gather tasks in parallel while passing data with each other through the interconnection network. Taking advantage of the ability, the next section examines and proposes several parallel processing techniques suitable for SPMD programs with completely-distributed/completely-related data references and a parallel computer architecture with an interconnection network.

Related work

A number of parallel algorithms for volume rendering have been proposed. Most algorithms can be categorized as direct ray-casting of volume data for MIMD/SIMD architectures[77]–[80]. In these algorithms, volume data is distributed among processing elements

and each processing element calculates the contribution of rays related to allocated subvolumes by making good use of pixel parallelism. Ray-casting, however, is computationally expensive in comparison with shear-warp factorization proposed in [72].

Recently, two parallel algorithms for shear-warp factorization have been proposed. Lacroute[81][82] proposed a parallel shear-warp factorization algorithm for two types of shared-memory parallel processing system (SGI Challenge and Stanford DASH). Lacroute achieved 15 frames/sec on an SGI Challenge 32-processing-element shared-memory system for $256^2 \times 167$ voxels. Nevertheless, this performance mainly relies heavily on both the high performance of the individual processing elements and the large cache memory in a Challenge system, and the scalability of this shared-memory implementation will be limited as the number of processing elements increases due to conflicts on shared resources such as shared-buses and shared-memories. This means that the distributed-memory parallel computer architecture is more suitable for data-parallel volume rendering as an SPMD program with completely-distributed/completely-related data references than the shared-memory one.

Amin et al.[83] proposed a parallel algorithm for shear-warp factorization for distributed-memory parallel processing systems. In their algorithm, the volume is first sheared and then partitioned by slicing perpendicular to volume slices. Subvolumes obtained by slicing are allocated to processing elements, and each processing element drives rays through its assigned volume segment. Since subvolume partition and allocation are performed after shearing, data communications are required whenever the shearing angle changes. Even though only the difference between the allocated subvolume data and needed one in changing the shearing angle is exchanged between neighboring processing elements, a

severe communication overhead arises for interactive operations that quickly change the view angle to a large volume data set. Consequently, the frame ratio is reduced and the changing of the view angle within a unit time is limited. These results describe the indispensability of reducing the data passed among processing elements, and of improving the transactions for data scattering and gathering in the SPMD programs.

To solve these problems, this chapter presents another data-parallel volume rendering algorithm including binary-swap compositing, merging with binary-tree communication and the adaptive volume subdivision. It is designed for the image construction system using data-parallel volume rendering as a distributed-memory parallel processing system with an interconnection network. In the following of this chapter, parallel shear-warp volume rendering including binary-swap compositing, merging with binary-tree communication and the adaptive volume subdivision is redefined as the parallel shear-warp volume rendering.

4.3 Parallel processing of an SPMD program with completely-distributed / completely-related data references on its parallel processing system

4.3.1 Data-parallel volume rendering with adaptive volume subdivision

If one scatter-gather task can be divided into some small scatter-gather tasks, each processing element can execute a small scatter-gather task using partial data while communicating with the others. In parallel shear-warp volume rendering, each of compositing and merging tasks is considered a set of the tasks. In the composition stage, a parallel task is the projection of subvolume images. A parallel task in the merge stage is the accumulation of

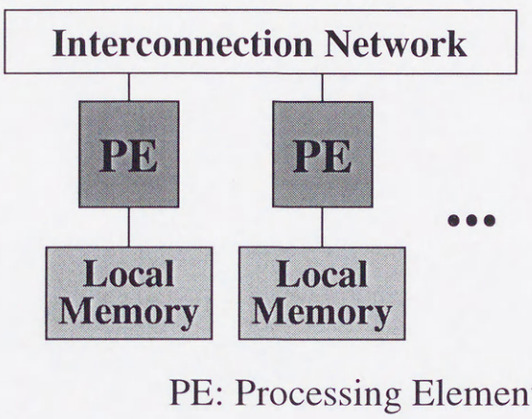


Figure 4.6: Parallel computer architecture for volume rendering.

partial final-images.

As improvements in data scattering and gathering for parallel shear-warp volume rendering, the binary-swap method[79] is employed to achieve fast image compositing. In addition, this section also proposes the adoption of binary-tree communication onto merge stage of the parallel shear-warp volume rendering.

If one scatter-gather task cannot be treated as a set of tasks, it is essential to reduce data passing among processing elements. This chapter proposes an adaptive volume subdivision for a reduction in data communication in the subvolume image composition while maintaining load balance among processing elements. Although the reduction technique for data communications is adopted for the image composition, the efficiency of the reduction is still applicable to general cases where one scatter-gather task cannot be treated

as a set of tasks.

Improvements in data scattering and gathering

To improve transactions for data scattering and gathering in parallel shear-warp volume rendering, the binary-swap method[79] and a communication scheme in a binary-tree fashion are employed into the composition stage and the merge stage of parallel shear-warp volume rendering, respectively.

– **Binary-swap compositing**

The binary-swap method was originally proposed for ray-cast volume rendering to locally integrate ray-cast subimages into a final image, and uses a bi-directional hypercube network among processing elements[85].

In the subvolume image compositing with the binary-swap method, called *binary-swap compositing* hereafter, is carried out as follows. Let p be the number of processing elements. In the first compositing step, two processing elements are paired only the least significant bits of their numbers are different (the Hamming distance between the two is 1). Processing elements of each pair exchange halves of their subvolume images. After exchanging, each processing element composites its half-image with the received half-image. At the n-th step, each processing element divides the newly composed image in the previous step into two half-images, each having a size of one-2^n-th of an original subvolume image. It sends one of half-images to the corresponding processing element whose number is different only in the n-th bit then. Each processing element composites the half-images as before. After $\log p$ steps, a partial image with a size of one-$2^{\log p}$ -th (i.e., one-p-th) of the intermediate

image will be held in each processing element.

Figure 4.7 shows an example of binary-swap compositing of subvolume images to the intermediate image over four processing elements.

As a result, binary-swap compositing gathers subvolume images that all processing elements have, composites the subvolume images, partitions the intermediate image and scatters the partitioned intermediate images consistently. Figure 4.8 describes the improved dependency of tasks by binary-swap compositing in the composition stage, in which the dependency by serial compositing is relaxed. Processing steps in the composition stage by binary-swap compositing takes $O(\log_2 p)$ time, while serial compositing which includes scattering of the intermediate image to all processing elements takes $O(2p)$.

Let T_{v-comp} be the time for compositing one voxel of each slice in a subvolume and A the area of a screen. Processing time of binary-swap compositing T_{comp} is obtained as follows:

$$\begin{aligned} T_{comp} &= \sum_{k=1}^{\log p} \{(T_{pass} + T_{v-comp})\frac{A}{2^k} + T_{setup}\} \\ &= A\ (T_{pass} + T_{v-comp})(1 - \frac{1}{p}) + T_{setup} \log p \end{aligned} \tag{4.1}$$

where T_{pass} and T_{setup} are the time for transmitting one pixel of an intermediate image between processing elements and the set-up time for one data-passing transaction, respectively. On the other hand, processing time by serial compositing $T_{serial-comp}$ is as follows:

$$\begin{aligned} T_{serial-comp} &= \sum_{k=1}^{p} (T_{setup} + T_{pass} + T_{v-comp})\frac{A}{p} + \sum_{k=1}^{p} (T_{setup} + T_{pass})\frac{A}{p} \\ &= A\ (2\ T_{setup} + 2\ T_{pass} + T_{v-comp}). \end{aligned} \tag{4.2}$$

These steps and processing times clearly show the advantage of binary-swap compositing.

Besides, the binary-swap method has another advantage that the image size for sending and compositing is effectively decreased while the parallel compositing proceeds. This makes the overall compositing process very efficient.

– Merging with binary-tree communication

After parallel warping, each processing element has a piece of a warped intermediate-image that is a final image to be displayed. These pieces of a final image distributed among processing elements must be gathered and merged to a single image. As described in Section 4.2.1, the gathering and merging are carried out by a merging task. This merging task is a scatter-gather task. The task can, however, be divided into some small merging-tasks, each of which partially gathers and merges a few pieces of a final image. The small merging-tasks should be executed in parallel. This involves that multiple communications among processing elements are also carried out in parallel.

Binary-tree communication is convenient for those task executions and communications. In addition, the gathering pieces of a final image is simply realized with binary-tree communication. Therefore, binary-tree communication is employed.

After $\log p$ binary-tree steps, a final image is gathered to a specific processing element and then displayed. Figure 4.9 illustrates the improvement of task dependency using binary-tree communication in merge stage upon the dependency by serial merging. Processing step of merge stage with binary-tree communication takes $O(\log_2 p)$ while serial merging takes $O(p)$.

The processing time for the merging with binary-tree communication T_{merge} is obtained as follows:

$$\begin{aligned} T_{merge} &= \sum_{k=1}^{\log p} \{T_{pass} \frac{A}{2^k} + T_{setup}\} \\ &= A\ T_{pass}(1 - \frac{1}{p}) + T_{setup}\ \log p. \end{aligned} \tag{4.3}$$

The processing time for serial merging $T_{serial-merge}$ is as follows:

$$\begin{aligned} T_{serial-merge} &= \sum_{k=1}^{p} (\frac{A}{p} T_{pass} + T_{setup}) \\ &= A\ T_{pass} + p\ T_{setup}. \end{aligned} \tag{4.4}$$

These equations describe that the processing times and steps can be reduced with binary-tree communication.

This section has manifested the effectiveness of the improvements in data scattering and gathering in parallel shear-warp volume rendering. Although these improvements are described by theoretical analysis, the analyses are still reliable for practical programs of parallel shear-warp volume rendering because the theoretical analyses are based only on the task dependencies in the composition stage and the merge stage.

As a result, the image construction system with an interconnection network, and improvements of the data scatter and gather transactions are valuable for parallel shear-warp volume rendering as an SPMD program with completely-distributed/completely-related data references.

Reduction of the data passed among processing elements

To reduce the amount of communications in the subvolume image composition with maintaining load balance among processing elements, an adaptive volume subdivision is proposed in this chapter.

As described above, each pair of processing elements exchanges and composites subvolume images in the composition stage. To carry out image composition efficiently, image data only within a minimum rectangle region (MRR), which contains all of effective pixels where opaque voxels are projected, should be exchanged and composited. An example of an MRR is shown in Figure 4.10. The amount of passed data is approximately proportional to the area of the MRR. Accordingly, the compositing time can be reduced by keeping the MRR small.

– **An adaptive volume subdivision**

Once the size and shape of subvolumes are determined as a result of volume subdivision, they are constant in a rendering process. Therefore, the size and shape of subvolumes can be adjusted to reduce the area of their MRRs in volume subdivision. To reduce the worst communication overhead, the proposed method adaptively subdivides a volume so that each MRR maximized by a specific viewing direction becomes as small as possible. As a result, subdivision is uniquely decided when volume data are given.

It is also important to balance loads in the parallel shear stage, which strongly influences the total processing efficiency. For this reason, the adaptive volume subdivision method gives priority to equalizing the number of opaque voxels of each subvolume. The

processing time of each processing element in the parallel shear stage tends to be approximately proportional to the number of opaque voxels in its subvolume. This means that the computational load of this stage can be balanced statically by equalizing the number of opaque voxels in each subvolume.

– A procedure for the adaptive volume subdivision

The proposed method recursively subdivides volume into two subvolumes by using any of the xy-, yz-, and zx-planes(primary planes). Here, an xy-plane is a plane perpendicular to the z-axis of the volume coordinate system. An appropriate subdividing plane is chosen to reduce the area of the maximized MRR and balance loads in the parallel shear stage as follows:

Step 1

Find the xy-, yz-, and zx-planes, each of which divides a volume into two subvolumes with the equal number of opaque voxels. Each plane generates two subvolumes (a pair of subvolumes).

Step 2

For each pair of the subvolumes obtained by one of the primary planes, find the subvolume generating a larger MRR than the other, called the larger-MRR subvolume. In consequence, three larger-MRR subvolumes related to the three primary planes are obtained.

step 3

Choose the plane that generates a subvolume with the smallest MRR among the three larger-MRR subvolumes for subdivision.

In Step 2, the area of each MRR maximized by a specific viewing direction has to be evaluated. Since precise evaluation of opacity distribution is computationally expensive, the area is approximately calculated by using a function derived from the bounding-box size of a subvolume.

Consider three principal axes of a volume space shown in Figure 4.12. The area-evaluation function for the z principal viewing direction is given as follows. A bounding box is supposed to contain s_z volume-slices with the same size of $s_x \times s_y$. These volume-slices are translated parallel to themselves so that viewing rays are parallel to the principal direction. When the MRR of the bounding box is maximized by a specific viewing direction, the distance between the front slice and the back slice in each of the x and y directions on the intermediate image plane is s_z. Accordingly, the maximum area of an MRR is calculated as:

$$A_z(s_x, s_y, s_z) = (s_x + s_z)(s_y + s_z). \tag{4.5}$$

For the x and y principal directions, the corresponding area-evaluation functions are similarly given as:

$$A_x(s_x, s_y, s_z) = (s_y + s_x)(s_z + s_x), \tag{4.6}$$

$$A_y(s_x, s_y, s_z) = (s_x + s_y)(s_z + s_y). \tag{4.7}$$

4.3.2 Performance evaluation

In this chapter, several experiments of the parallel shear-warp volume rendering are carried out to discuss the parallel computer architecture and the parallel processing techniques for SPMD programs with completely-distributed/completely-related data references.

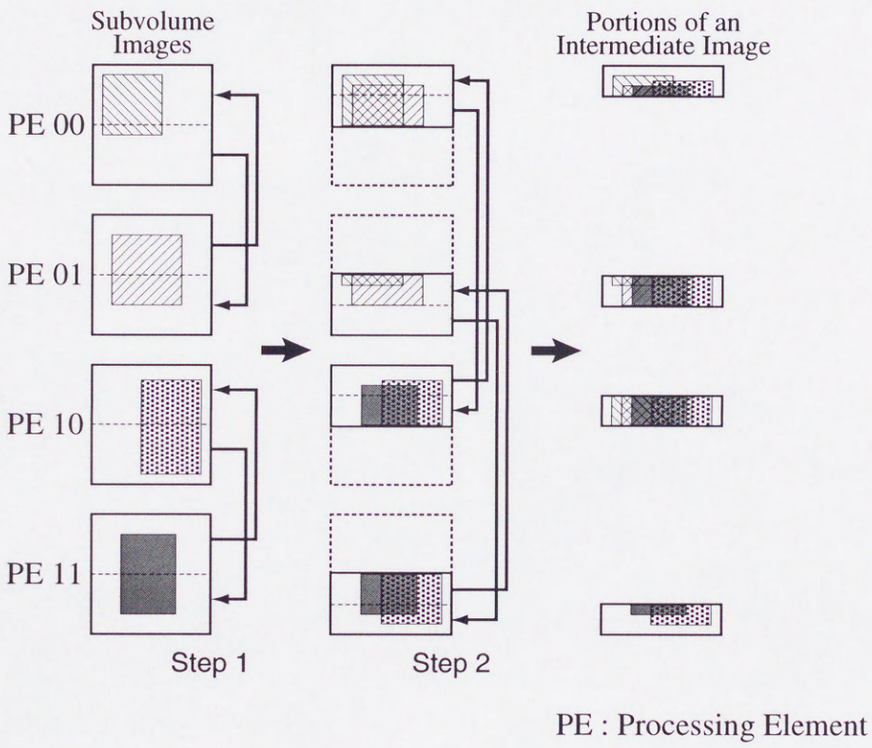


Figure 4.7: Binary-swap compositing for intermediate image generation.

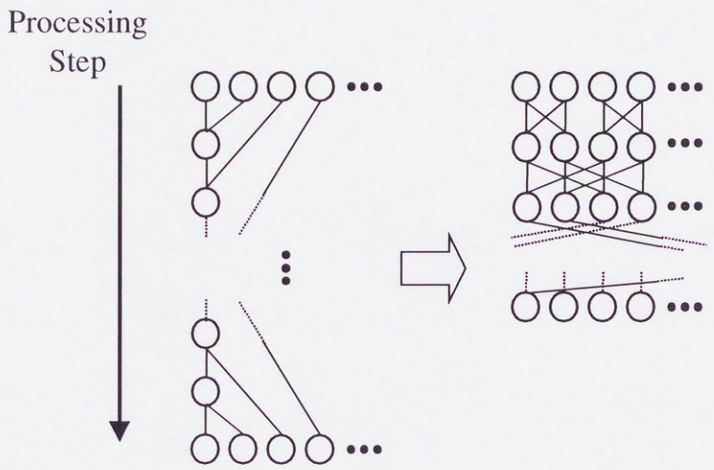


Figure 4.8: Task dependency by binary-swap compositing in the composition stage.

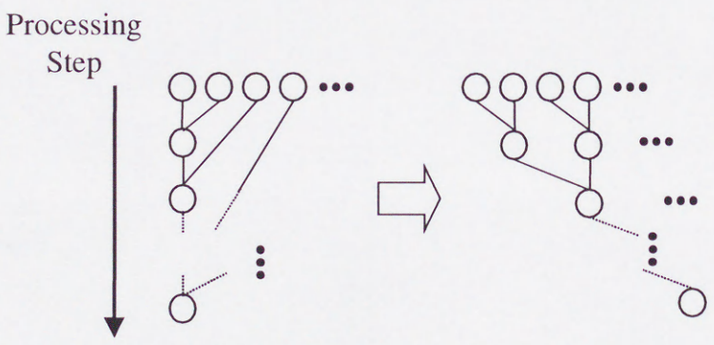


Figure 4.9: Task dependency using binary-tree communication in the merge stage.

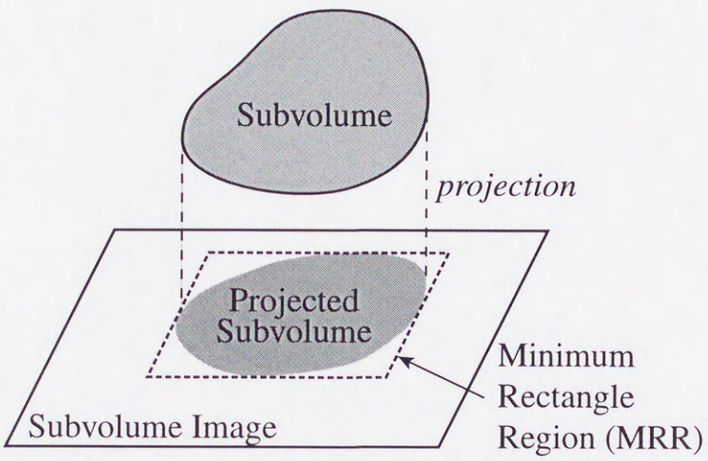
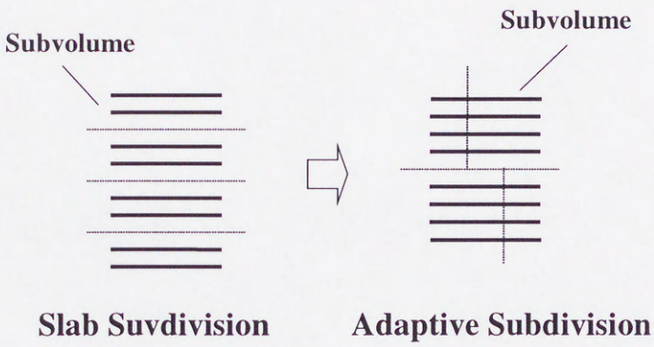


Figure 4.10: An example of a Minimum Rectangle Region(MRR).

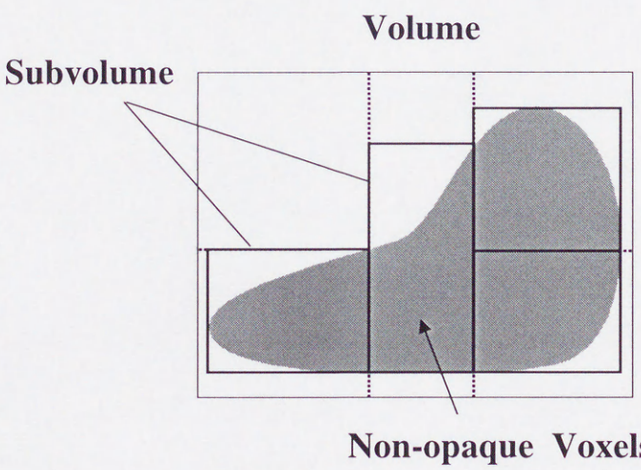
The effectiveness of the improvements for data scattering and gathering in parallel shear-warp volume rendering has already been indicated in the previous section. Therefore, the experiments are concerning the proposed adaptive volume subdivision.

Environment for experiments

This chapter implements the parallel shear-warp volume rendering on an IBM SP2[86] as the image construction system. The IBM SP2 has 32 nodes as processing elements, each of which is RISC System/6000 POWER2 node[87] operating at a 15 ns clock. Each node has a 32 KB instruction cache, 64 KB data cache, 64-bit memory bus, and 128-bit bus between the data cache and the FPU (a 2-port bus). The network subsystem that



(a) Subvolume.



(b) Intermediate image.

Figure 4.11: An example of adaptive volume subdivision.

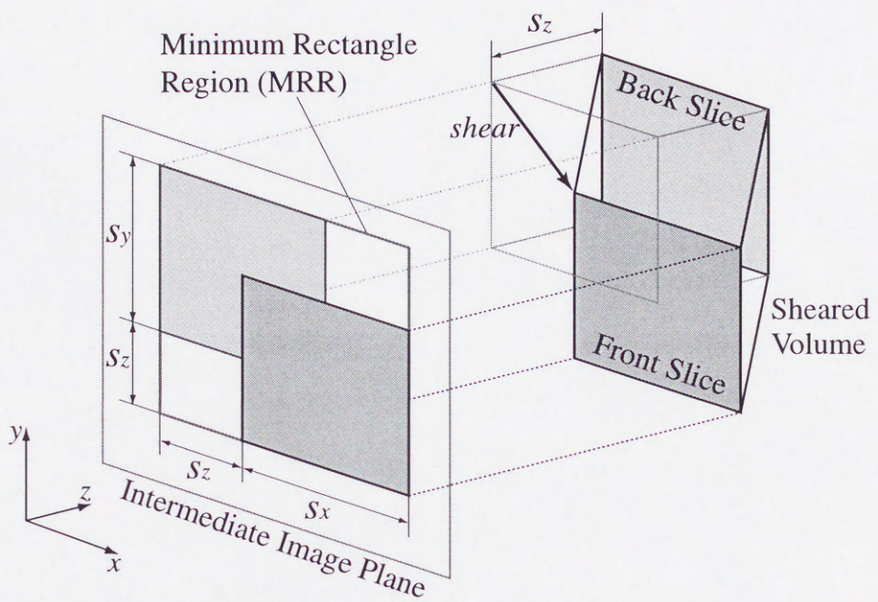


Figure 4.12: The relationship between a sheared volume and its MRR on the intermediate image plane.

interconnects 32 POWER2-nodes is based upon a low-latency, high-bandwidth interconnection network called the High Performance Switch (HPS). The HPS is a bi-directional multistage interconnection network that scales the bisection bandwidth linearly with the number of nodes while maintaining a fixed number of communication ports per processor node. The HPS operates at 40 MHz, providing a peak bandwidth of 40 MB/sec in each direction.

The implementation is described in C++ with the MPL[88] message-passing library. MPL is IBM's proprietary message-passing library, but it is quite similar to the MPI[89]–[91] message-passing library.

The experiments also use gray-scale images with the size of 256^2 pixels, from two data sets: *skull* (256^3 voxels from CT (Computer Tomography) and *head* ($256^2 \times 128$ voxels from MR (Magnetic Resonance) scan). Figures 4.13 (a) and (b) show test images rendered with the data sets. These data sets are included in the volume rendering test data sets produced by the Chapel Hill, and they are scaled into the appropriate size. After classification, 12.1% and 6.1% of all voxels are classified into opaque voxels for *Head* and *Skull*. Shading, classification, and subdivision of the volume data set are carried out as preprocessing.

Since the time for each stage also depends on viewing parameters, two viewing directions, which is view1 and view2 shown in Figure 4.14, are considered in the experiments. View2 would result in the longest time in the composition stage because the area of an MRR was maximized by the viewing direction, while view1 results in the shortest time. Besides, in the experiments, the adaptive volume subdivision is compared with a simple

slab subdivision that was employed in [24]. Figure 4.15 illustrates an example of slab subdivision. In slab subdivision, a volume is recursively subdivided into 2^n subvolumes by planes parallel to one of the primary planes in a volume space, which is the most perpendicular to the viewing direction. Each of the subdividing planes divides a (sub)volume into two subvolumes, which have the same number of opaque voxels for static load balancing.

As other parameters, the experiments are carried out on 1, 2, 4, 8, 16 and 32 processing elements (nodes) of the SP2. Data exchange and calculations between paired processing elements in the composition and merge stages are simultaneously performed. The serial merging is employed in the experiments because the gathering and accumulating pieces of a final image in the merge stage need much less time than the processing times in the other stages.

Experimental results

The experiments are carried out to examine the processing time of each stage and total processing time of volume rendering.

Figures 4.16 (a) and (b) show the timing results in the composition stage for view1 and view2 with *skull* and *head*, respectively. In each case, the processing time for view2 is about two times longer than that of view1 because the MRR caused by view2 is bigger than that by view1.

Figures 4.16 (a) and (b) depict that the compositing time with the slab subdivision method almost always increases as the number of processing elements increases even

(a) Skull.

(b) Head.

Figure 4.13: Test images from Skull and Head data sets.

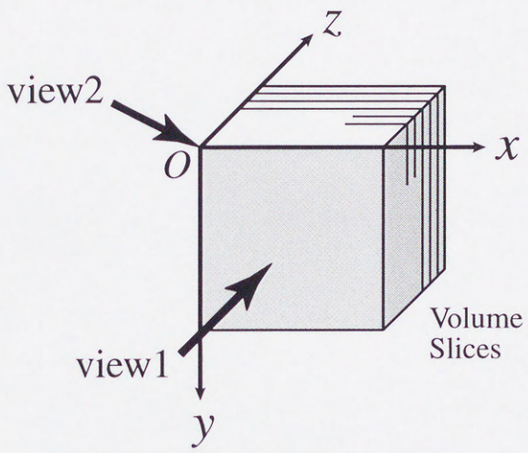


Figure 4.14: Two views for experiments: view1=(0,0,1), view2=(1,1,1).

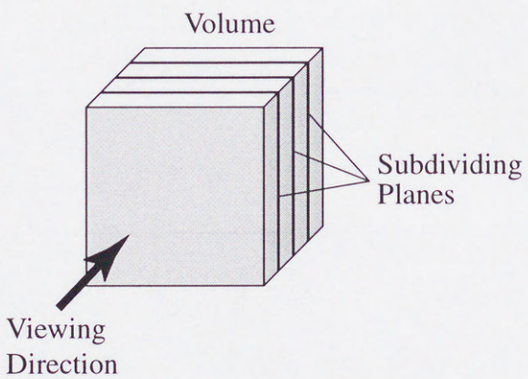


Figure 4.15: An example of slab subdivision.

though the increasing rate declines. This tendency means that the area of composited regions of subvolume images hardly decreases as the number of processing elements increases in the slab subdivision. On the other hand, the compositing time by the adaptive subdivision tends to decrease for both of view1 and view2 when the number of processing elements exceeds 8. This manifests that the MRRs become smaller as the number of subdivisions increases in the adaptive subdivision. As a result, in the case of 32 processing elements, the compositing time obtained by adaptive subdivision is almost half of that by using slab subdivision.

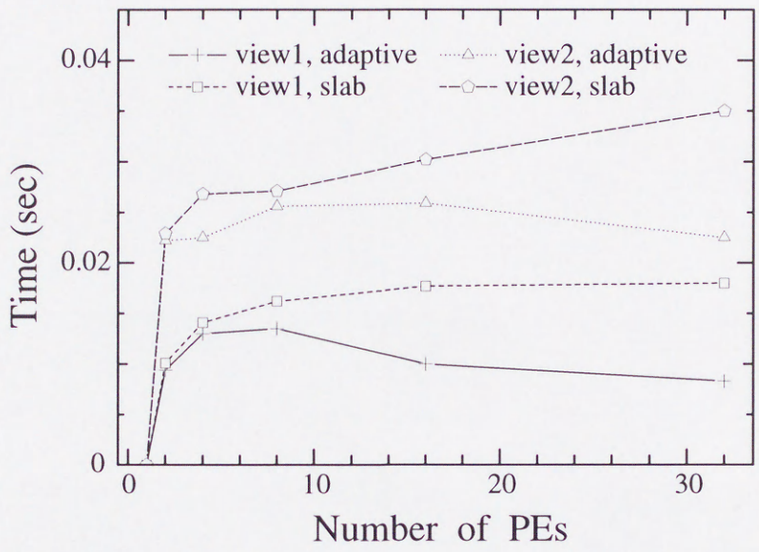
For view2, however, adaptive subdivision does not always result in shorter compositing time compared with slab subdivision. This is caused by differences in location of MRRs because not only the area of MRRs but also their locations on the intermediate image plane affect the total amount of image data exchanged in the composition stage. The area of MRRs by adaptive subdivision becomes smaller than that by slab subdivision as the number of processing elements increases. Accordingly, the effect of MRR locations is trivial compared with the effect of the decreased area of MRRs. This presents that the more processing elements are employed, the much shorter compositing time is achieved by adaptive subdivision compared to slab subdivision.

Figure 4.17 illustrates speedups in the parallel shear stage. There is only a little difference in speedups with regard to volume data sets and kinds of subdivision. In this stage, the number of opaque voxels in each subvolume affects the performance. The high scalability in the parallel shear stage shown in Figure 4.17 means that this stage is effectively parallelized.

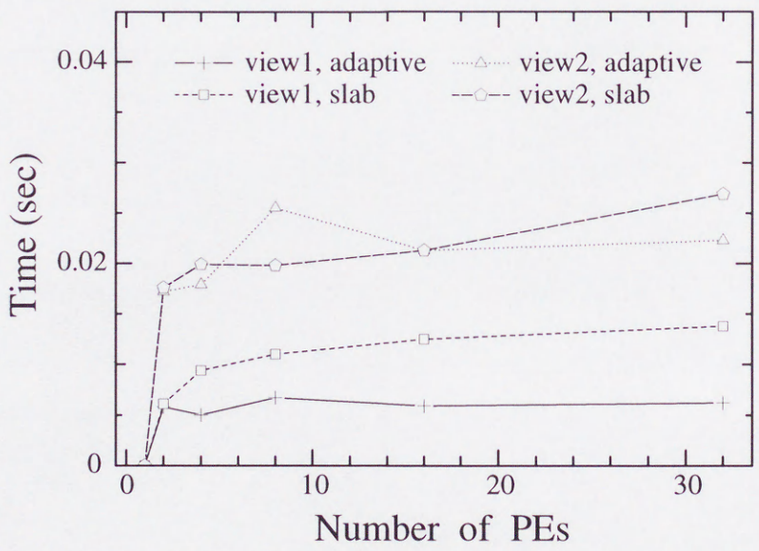
Figure 4.18 shows the number of opaque voxels in each subvolume and the load balancing effect in the parallel shear stage obtained by adaptive volume subdivision. This figure represents that the number of opaque voxels in each subvolume and the processing time of the parallel shear stage are balanced well for both *skull* and *head* data sets.

Figure 4.19 shows the processing-time ratio of each stage to the total time in the case of view2. As illustrated in this figure, the processing-time ratio in the composition stage to the total time increases as the number of processing elements increases. This means that the processing time of the composition stage becomes dominant as the number of processing elements increases. The total speedup of parallel shear-warp volume rendering for view2 is shown in Figure 4.20. Adaptive volume subdivision explicitly affects the total speedups. Adaptive subdivision achieves a lower ratio in the composition stage and higher speedups than slab subdivision in each figure, respectively.

As a consequence, the adaptive volume subdivision method proposed in this thesis has the high effectiveness in parallel shear-warp volume rendering. This indicates the indispensability of reducing the amount of communicating data among processing elements with maintaining the load balance among them for SPMD programs with completely-distributed/completely-related data references.



(a) Skull.



(b) Head.

Figure 4.16: Processing time of the composition stage.

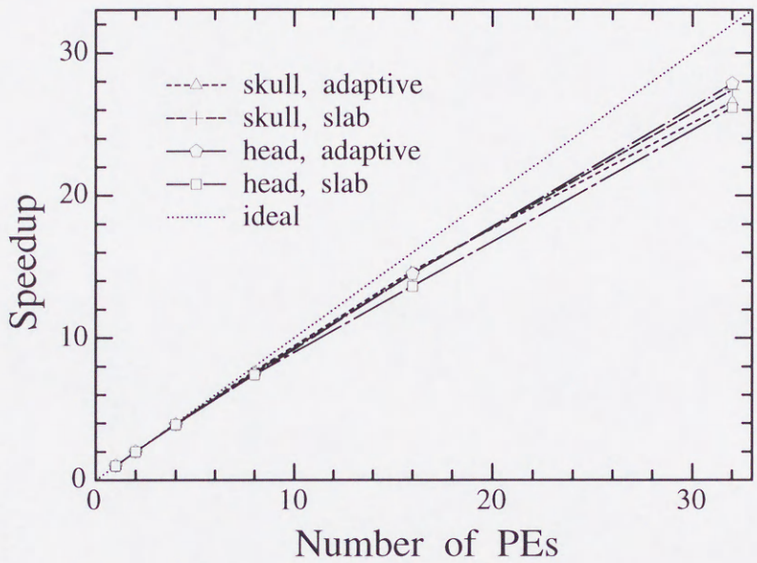


Figure 4.17: Speedups of the parallel shear stage.

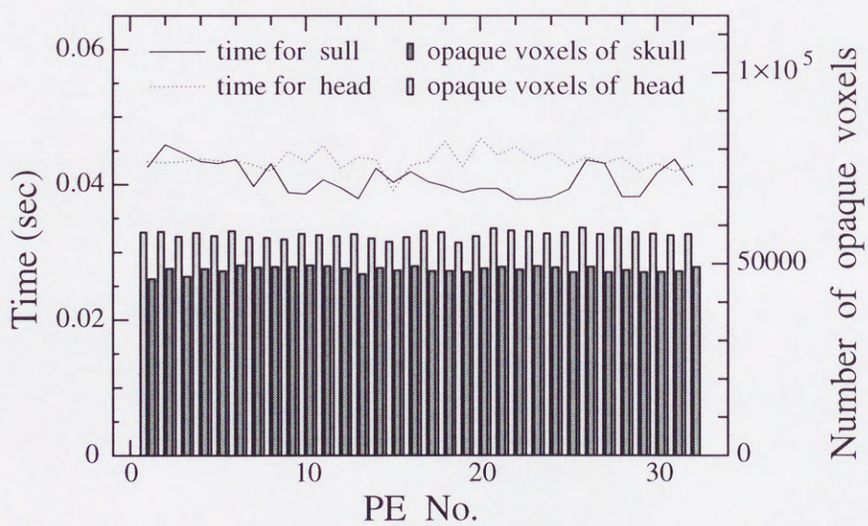


Figure 4.18: Effect of load balancing on performance in the parallel shear stage.

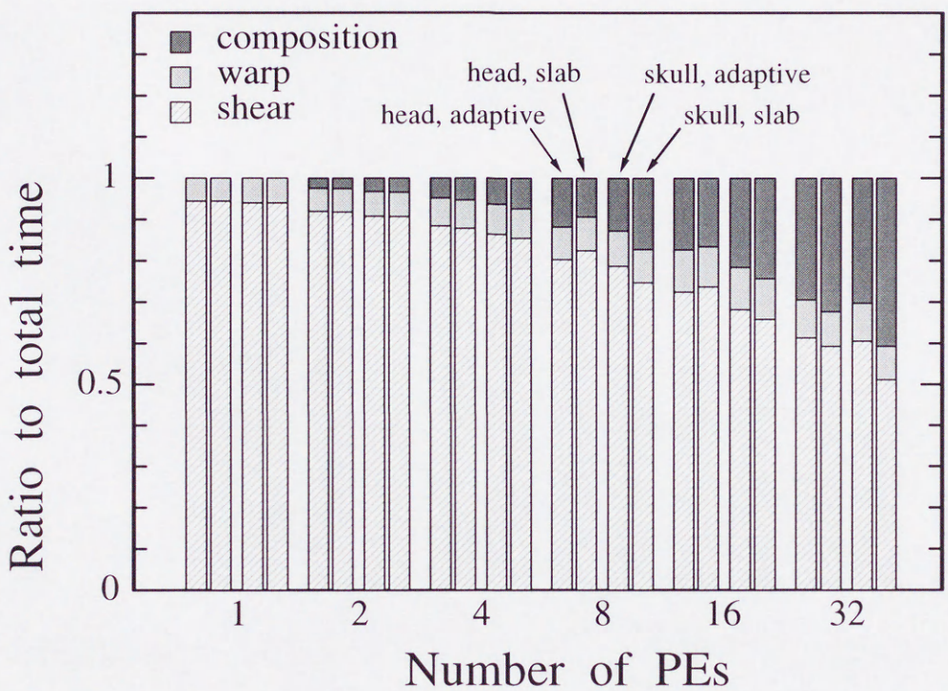
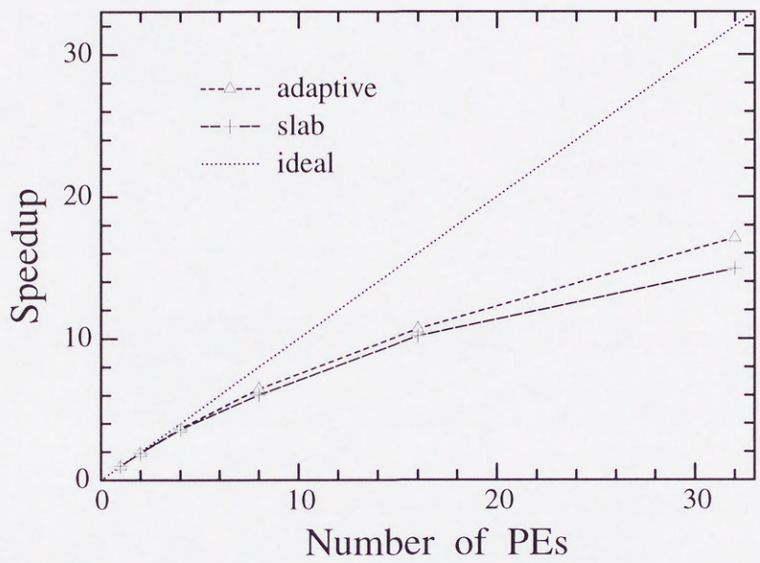
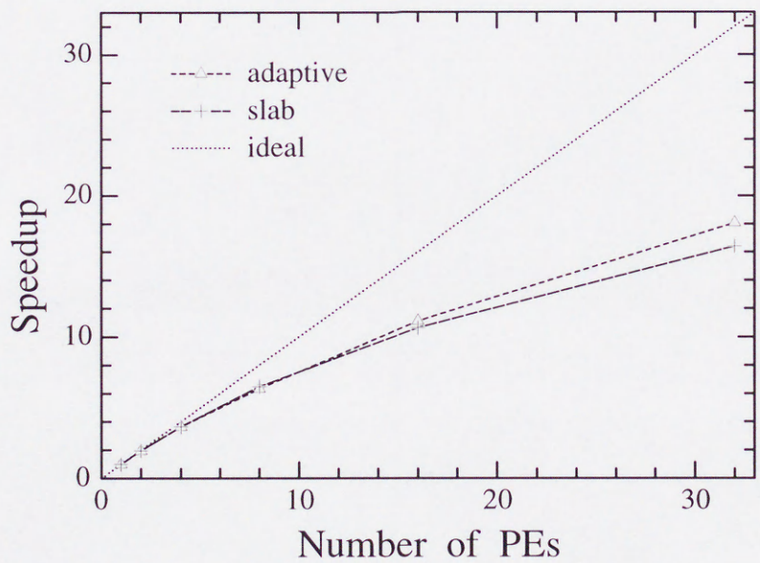


Figure 4.19: Breakdown of the total processing time.



(a) Skull.



(b) Head.

Figure 4.20: Total speedups.

4.4 Conclusions

This chapter has discussed about data-parallel volume rendering as SPMD programs with completely-distributed/completely-related data references, and described an image construction system, which is a distributed-memory parallel processing system with an interconnection network, to achieve high-speed parallel processing of volume rendering.

This chapter also has applied the binary-swap method to image compositing and proposed the adaptive volume subdivision method so that the processing time of data communication can be reduced and the load can be balanced among processing elements. Moreover, this chapter has discussed the parallel processing system and proposed techniques through some experiments.

The experimental results has shown, the effectiveness of a distributed-memory parallel processing system with an interconnection network and the requisiteness of techniques to decrease the processing time taken by data communication (i.e., the improvements of data scattering and gathering transaction) and the reduction of the data passed among processing elements. This implies that efficient data communication among processing elements is more intrinsic than high-speed processing of parallel tasks for the parallel processing of the SPMD programs with completely-distributed/completely-related data references.

5 A Parallel Computer Architecture for SPMD Programs with Completely-Distributed / Partially-Related Data References

5.1 Introduction

Regarding SPMD programs with completely-distributed/partially-related data references described in Chapter 2, this chapter discusses the effectiveness of the distributed-memory parallel computer architecture with an interconnection network and the indispensability of selecting the proper data-reference locality to achieve high-speed parallel processing of the programs.

This chapter focuses my attention on active contour models (ACMs)[93] as an example of SPMD programs with completely-distributed/partially-related date references. ACM is one of image processing techniques, which uses a deformable curve for extracting a region-of-interest (ROI) from various images such as natural images and medical images. ACM has, however, the problem that efficient parallel processing of the program is difficult to be achieved because of its strict dependency of tasks.

At the beginning, this chapter briefly describes ACM and a distributed-memory parallel processing system with an interconnection network suitable for programs of ACM. In addition, another active contour model with locally considering region-of-interest shape

is proposed in this chapter[25][26]. Programs of the proposed ACM can be executed in parallel utilize the partial dependencies of tasks[25]. Besides, this chapter discusses effectiveness of the system and importance of sufficiently utilizing the local dependencies of tasks through some experiments on parallel processing of the program.

5.2 An SPMD program with completely-distributed / partially-related data references and its parallel processing system

5.2.1 Region-of-interest extraction using an active contour model

An active contour model (ACM)[93] is one of image processing techniques, which uses a deformable curve for extracting a region-of-interest (ROI) contour from source images such as natural images and medical images. ACM is also known as a physically based technique using smoothness restrictions and external forces to find out an ROI contour. In some constrained case, this model has the effectiveness to simultaneously solve both the segmentation and tracking problems; e.g., extracting ROI that is lacking the partial information of its contour, tracking ROI contour on a noisy image and obtaining ROI that has a particular shape. The advantage of ACMs attracts many researchers to these problems[94]–[98]. Extension of ACMs toward 3D or timing-space is also studied by some researchers[99]–[103].

A program based on ACM is, however, time consuming and has the problem that the efficient parallel processing to speedup the program is difficult to be achieved, even though some techniques for high-speed processing of the program on a single processing-element system have been reported[104]–[109] and used in many models[96][97][101][102][110]–[112].

Active contour model

ACM uses a deformable curve to extract an ROI contour[93]. The deformable curve is given as an initial contour by users, and repeats the renewal of its position to find out an ROI contour. Figure 5.1 illustrates an overview of the ROI extraction with an active contour model. This is a reason why the deformable curve is called an *active contour*. Kass et al. also named the deformable curve a *snake*[93].

Consider an active contour represented by $\{v(s,t) = ((x,(s,t),y(s,t)) : s \in \Omega, t \in T\}$, where parameter s is a spatial index on given an open interval Ω and parameter t is time defined on T. A portion of an active contour represented as $v(s,t)$ is called a *partial contour*. In other words, an active contour consists of partial contours. Although v is a vector, this chapter denotes it by using v instead of $\mathbf{v}$ for simplicity.

The potential energy of an active contour at each time is defined as follows[93]:

$$E_{snake} = \frac{1}{2}\int_{\Omega}[E_{int}(v(s)) + E_{field}(v(s)) + E_{con}(v(s))]ds \tag{5.1}$$

where E_{int}, E_{field}, and E_{con} represent the internal potential energy, the potential energy that causes the field force, and the energy generating the external constraint force, respectively. A potential energy is defined for each partial contour and E_{snake} is obtained by integrating all the potential energies as described in Eq.5.1. An active contour extracts an ROI contour from a source image by finding the minimum value of E_{snake}.

Each potential energy $E_{int}(v(s))$, $E_{field}(v(s))$, and $E_{con}(v(s))$ of a partial contour $v(s)$ corresponds to a particular force that works on the partial contour.

- E_{int} : the internal potential energy
  - A potential energy causes the internal force $\vec{f}_{int}$.
  - A partial contour resists bending and stretching of a contour segment, which includes the partial contour, with $\vec{f}_{int}$.
- E_{field} : the field potential energy
  - A potential energy generates a field force $\vec{f}_{field}$.
  - A partial contour is pulled or pushed by $\vec{f}_{field}$ with the attributes of a source image such as a pixel value, a gradient of a pixel value, and texture of a source image.
- E_{con} : the external-constraint potential energy
  - A potential energy creates an external constraint force $\vec{f}_{con}$.
  - $\vec{f}_{con}$ is defined so that a partial contour travels on a source image according to constraints intended by users.

This chapter hereafter uses f instead of $\vec{f}$ for simplicity.

The equation-of-motion of an active contour is derived from Eq.5.1 and an active contour travels under the equation-of-motion.

Moreover, the equation-of-motion is discretized in two domains; i.e., space and time to be numerically solved. The timing discretization of the equation-of-motion means that an active contour moves at a certain interval. On the other hand, the spatial discretization indicates that an active contour as a continuous function is converted into a set of contour

points as a discretized function. Since the discretized equation-of-motion calculates the position of each partial contour at a moment, the equation-of-motion of a partial contour is regarded as an executing unit of an ACM program; i.e., a task. Equation5.1 denotes that a task should include not only the equation-of-motion of a partial contour $v(s)$ but also the execution of the three potential energies; i.e., $E_{int}(v(s))$, $E_{field}(v(s))$ and $E_{con}(v(s))$.

The following subjects explain the equation-of-motion of an active contour and the discretization of the equation to specify the dependency of tasks in an ACM program.

The equation-of-motion of an active contour

In general, the internal potential energy E_{int} at time t is indicated as follows[93][94]:

$$E_{int}(v(s,t)) = \omega_1(s)\ |v_s(s,t)|^2 + \omega_2(s)\ |v_{ss}(s,t)|^2 \tag{5.2}$$

using the first-order and second-order differential of $v(s,t)$ by s where $v_s \equiv \partial v/\partial s$ and $v_{ss} \equiv \partial^2 v/\partial s^2$. In Eq.5.2, $\omega_1(s)$ and $\omega_2(s)$ are weights in each term of $v(s,t)$. The first-order term $\omega_1(s)\ |v_s(s,t)|^2$ and the second-order term $\omega_2(s)\ |v_{ss}(s,t)|^2$ make an active contour resist stretching and bending, respectively. The flatter the segment of an active contour including a partial contour $v(s)$ is, the smaller the internal potential energy $E_{int}(v(s))$ is.

$f_{int}(v(s,t))$ is derived with the variational derivation of E_{int} as follows[93][94][113]:

$$f_{int}(v(s)) = -\frac{1}{2}\frac{\delta E_{int}(v(s))}{\delta v(s)} = -\frac{\partial}{\partial s}\left(\omega_1(s)\ v_s(s)\right) + \frac{\partial^2}{\partial^2 s}\left(\omega_2(s)\ v_{ss}(s)\right). \tag{5.3}$$

E_{int} (f_{int}) gives an active contour the smoothness. By contrast, E_{field}, and E_{con} (f_{field}, and f_{con}) are arbitrarily defined by the user as a function of v to characterize the

active contour.

The Euler-Lagrange equation-of-motion is obtained from Eq.5.1, 5.2, and 5.3 as follows:

$$\begin{aligned} & \mu v_{tt}(s) + \gamma v_t(s) - \frac{\partial}{\partial s}(\omega_1(s)\ v_s(s)) + \frac{\partial^2}{\partial^2 s}(\omega_2(s)\ v_{ss}(s)) \\ = \ & f_{con}(v(s)) + f_{field}(v(s)) \end{aligned} \tag{5.4}$$

where $v_t \equiv \partial v/\partial t$ and $v_{tt} \equiv \partial^2 v/\partial t^2$. An active contour travels on a source image under the equation-of-motion of Eq.5.4.

Discretization of the equation-of-motion

The spatial discretization of Eq.5.4 stands for the sampling an active contour with N points: $\{\bar{s} : \bar{s} \in \bar{\Omega}, \bar{\Omega} = \{1, \ldots, N\}\}$. Accordingly, $v(s)$ is replaced with $\bar{v}(\bar{s})$. Equation5.4 is rewritten into:

$$\mathbf{M}\mathbf{V}_{tt}(\bullet) + \mathbf{C}\mathbf{V}_t(\bullet) + \mathbf{K}\mathbf{V}(\bullet) = \mathbf{R}_{\bar{v}}(\bullet) \tag{5.5}$$

where $\bullet$ means all contour points: $\mathbf{V}(\bullet) = (\bar{v}(1), \bar{v}(2), \ldots, \bar{v}(N)^T$. $\mathbf{M}$ and $\mathbf{C}$ are the mass and damping matrices, respectively. $\mathbf{M}$ and $\mathbf{C}$ are diagonalized matrices: $\mathbf{M} = \mu\mathbf{I}$ and $\mathbf{C} = \gamma\mathbf{I}$ where $\mathbf{I}$ is the identity matrix. $\mathbf{R}_{\bar{v}}$ is a matrix of external restriction forces (external load forces) that consist of the field force and external constraint force : $R_{\bar{v}}(\bar{v}(\bar{s})) = f_{con}(\bar{v}(\bar{s})) + f_{field}(\bar{v}(\bar{s}))$.

$\mathbf{K}$ is known as the stiffness matrix, and is derived from $f_{int}(\bar{v}(\bar{s}))(= -\frac{1}{2}E_{int\ \bar{v}}(\bar{v}(\bar{s})))$:

$$\mathbf{K} = \begin{pmatrix} c_1 & b_1 & a_1 & & & & a_{N-1} & b_N \\ b_1 & c_2 & b_2 & a_2 & & \mathbf{0} & & a_N \\ a_1 & b_2 & c_3 & b_3 & a_3 & & & \\ & a_2 & b_3 & c_4 & b_4 & a_4 & & \\ & & & & \ddots & & & \\ & & & a_{N-4} & b_{N-3} & c_{N-2} & b_{N-2} & a_{N-2} \\ a_{N-1} & & \mathbf{0} & & a_{N-3} & b_{N-2} & c_{N-1} & b_{N-1} \\ b_N & a_N & & & & a_{N-2} & b_{N-1} & c_N \end{pmatrix} \tag{5.6}$$

using the following $a_{\bar{s}}$, $b_{\bar{s}}$, and $c_{\bar{s}}$:

$$\begin{aligned} a_{\bar{s}} &= \frac{1}{h^4(\bar{s})}\ \omega_2(\bar{s}+1), \\ b_{\bar{s}} &= \frac{1}{h^4(\bar{s})}\ \Big(-2\ \omega_2(\bar{s}) - 2\ \omega_2(\bar{s}+1) - h^2(\bar{s})\ \omega_1(\bar{s}+1)\Big), \\ c_{\bar{s}} &= \frac{1}{h^4(\bar{s})}\ \Big(\omega_2(\bar{s}-1) + 4\ \omega_2(\bar{s}) + \omega_2(\bar{s}+1) \\ &\qquad + h^2(\bar{s})\ \omega_1(\bar{s}) + h^2(\bar{s})\ \omega_1(\bar{s}+1)\Big), \end{aligned} \tag{5.7}$$

where $h(\bar{s})$ is distance between $\bar{v}(\bar{s})$ and $\bar{v}(\bar{s}+1)$. Equation5.6 indicates that $\mathbf{K}$ is symmetric and pentadiagonal[94][113].

Next, Eq.5.5 is discretized in the time domain as follows:

$$\begin{aligned} &\left(\frac{1}{(\Delta\bar{t})^2}\mathbf{M} + \frac{1}{(2\ \Delta\bar{t})}\mathbf{C}\right)\ \mathbf{V}(\bullet, \bar{t}) \\ =\ &\mathbf{R}_{\bar{v}}(\bullet, \bar{t} - \Delta\bar{t}) + \left(\frac{2}{(\Delta\bar{t})^2}\mathbf{M} - \mathbf{K}\right)\ \mathbf{V}(\bullet, \bar{t} - \Delta\bar{t}) \\ &+ \left(-\frac{1}{(\Delta\bar{t})^2}\mathbf{M} + \frac{1}{(2\ \Delta\bar{t})}\mathbf{C}\right)\ \mathbf{V}(\bullet, \bar{t} - 2\ \Delta\bar{t}) \end{aligned} \tag{5.8}$$

by using the central difference formulae in terms of $\bar{t} - \Delta\bar{t}$:

$$\begin{aligned} v_t(\bar{s}, \bar{t} - \Delta\bar{t}) &\approx \frac{1}{2\Delta\bar{t}}[\bar{v}(\bar{s}, \bar{t}) - \bar{v}(\bar{s}, \bar{t} - 2\Delta\bar{t})], \\ v_{tt}(\bar{s}, \bar{t} - \Delta\bar{t}) &\approx \frac{1}{(\Delta\bar{t})^2}[\bar{v}(\bar{s}, \bar{t}) - 2\bar{v}(\bar{s}, \bar{t} - \Delta\bar{t}) + \bar{v}(\bar{s}, \bar{t} - 2\Delta\bar{t})]. \end{aligned} \tag{5.9}$$

Equation5.8 specifies that the equation-of-motion of each contour point needs the coordinate values of four neighboring points that contain two neighboring points in each side. Figure 5.2 shows the dependency among contour points. As shown in Figure 5.2, $\bar{v}(i, \bar{t})$ depends on $\bar{v}(i-2, \bar{t}-\Delta\bar{t}) \sim \bar{v}(i+2, \bar{t}-\ \Delta\bar{t})$ and $\bar{v}(i, \bar{t}-2\ \Delta\bar{t})$. Hereafter, although $\bar{v}$ is a contour point as a discretized contour, this chapter denotes it using v instead of $\bar{v}$ for simplicity.

In the parallel processing of an ACM program, a task that consists of the equation-of-motion and the potential-energy execution for a contour point can be allocated to a processing element. This is because the tasks of an ACM program can be executed in parallel. The dependency of contour points, however, oblige a processing element treating a task to communicate with processing elements that execute the tasks of its four neighboring points. Figure 5.3 depicts the task dependency during the program processed. As a consequence, an ACM program is regarded as an SPMD program with completely-distributed/partially-related data references.

5.2.2 An image processing system with an active contour model for region-of-interest extraction

Regarding SPMD programs with completely-distributed/partially-related data references, perfect-parallel processing of tasks is accomplished when the data references are completely distributed similarly in the case of the completely-distributed/completely-related type. In addition, a number of local data passing among processing elements can be treated in parallel. Therefore, this chapter discusses the system based on the distributed-memory parallel computer architecture with an interconnection network as an image processing

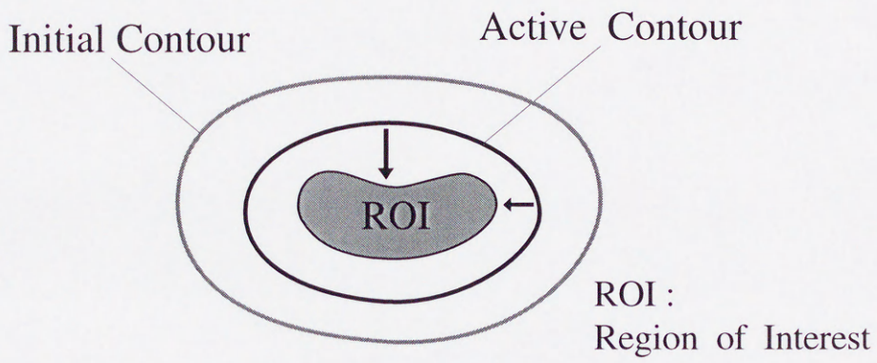


Figure 5.1: Active Contour Model.

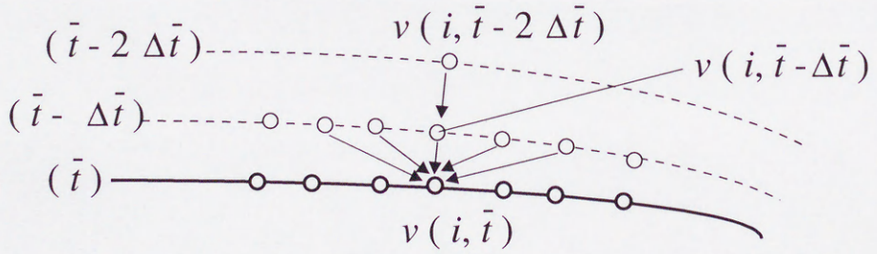


Figure 5.2: Dependency among contour points.

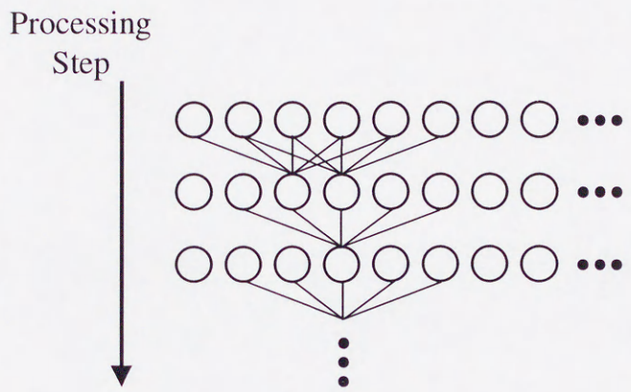


Figure 5.3: Task dependency of active contour model.

system for ACM, through which multiple communications can be treated among processing elements. The block diagram of the system in Figure 5.4 is the same as the one for the completely-distributed/completely-related type of SPMD programs in Chapter 4(Figure 5.4).

Consider that multiple tasks are allocated into a processing element. Although an ACM program is an SPMD program with completely-distributed/partially related data references, all tasks that communicate with each other cannot be allocated into the same processing element because of their dependencies. Each processing element has to pass a part of the data obtained by executing tasks of its own to other processing elements every time-step. It takes a certain time for the communication according to the number of allocated tasks. In this case, after preferentially executing the tasks whose results are to be passed, each processing element may execute the remaining tasks and communicate

with other processing elements simultaneously. The simultaneous treatment of the task execution and communication would be valuable for the parallel processing of the program. It must, however, be noted that the allocation of multiple tasks gives rise to a tradeoff between the parallel-processing efficiency with hiding the communication latency and high-speed parallel processing of the program.

Related work

An ACM program is time consuming, and has been thought that its efficient parallel processing based on contour points is difficult to be accomplished because of the task dependencies. Actually, although the tasks of an ACM program can be executed in parallel based on contour points as described in Eq. 5.8, the parallelization of ACM programs still has not been proposed, while some techniques for high-speed processing of the program on a single processing-element system have been reported; e.g., greedy algorithm[106][107], dynamic programming[104][105][109] and the model using a Hopfield network[108]. These techniques are used in many ACMs[96][97][101][102][110]–[112]. Unfortunately, in these models, the forces f_{field} and/or f_{con} of a contour point have often been defined using global information of an active contour such as locations of all contour points. The utilization of the global information constrains ACM programs to be treated as SPMD programs with SPMD programs with completely-distributed/completely-related data references.

The parallelization of ACM programs based on contour points must be essential for accelerating their processing when ACMs are applied to high-resolution images (e.g., medical images) with large amount of contour points. Parallel processing of an active contour

in terms of contour points is discussed with proposing an ACM that uses local shapes of ROI. In the proposed ACM, the forces f_{field} and f_{con} of a contour point are defined within the local dependencies of tasks restricted by the definition of f_{int}. Therefore, the programs of the proposed ACM are regarded as an SPMD program with completely-distributed/partially-related data references.

The proposed ACM with considering the local shapes of ROI is similar to ACMs utilizing the information of ROI shapes reported by some researchers; e.g., CARM-snake using complex autoregressive model[95] and M-snake with a sample contour model[96]. However, these models are not intended to be parallelized. The elastic contour model also uses the information of ROI shapes[114]. Since the elastic contour model does not deal with the equation-of-motion of an active contour, the model is not an active contour model (ACM). In addition, the parallelization employed in the elastic contour model is applied not to contour points of a contour but to candidates for an ROI contour. Hence, programs based on the elastic contour model are not regarded as SPMD programs with completely-distributed/partially-related data references to be discussed.

The next section proposes an ACM that uses the local shapes of ROI to discuss the parallel processing of ACM programs based on contour points. After that, the utilization and selection of proper data-reference locality in ACM program are examined through some experiments on the parallel processing of the program with the image processing system.

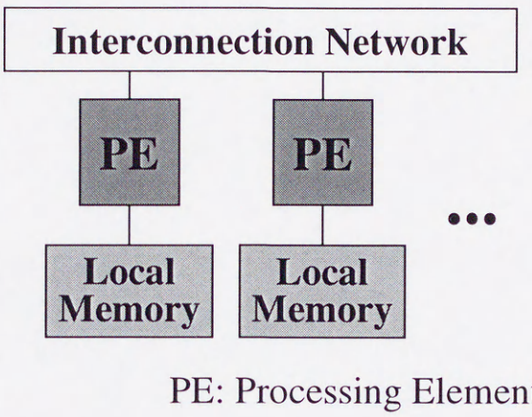


Figure 5.4: Parallel computer architecture for an active contour model.

5.3 Parallel processing of an SPMD program with completely-distributed / partially-related data references on its parallel processing system

5.3.1 An active contour model using local shape information of a region-of-interest

It is important to exploit the data-reference locality in an SPMD program with completely-distributed/partially-related data references. Not sufficiently utilizing the data-reference locality obviously increases communications among processing elements.

At the beginning, to discuss the utilization of data-reference locality, this section redefines the equation-of-motion of an active contour so as to be suitable for the completely-distributed/partially-related type of SPMD programs. After that, the selection of proper data-reference locality, which would be essential for the parallel processing of the SPMD

program, is examined for a program of the proposed ACM.

The field potential energy and the field force

In an ACM program, the equation-of-motion of a contour point needs the coordinate values of the four neighboring contour points because of the task dependency specified by f_{int} in Eq.5.8. Accordingly, E_{field}, f_{field}, E_{con} and f_{con} have to be calculated with these four neighboring points. E_{field} and f_{field} are defined by using coordinate values of at most the four neighboring contour points so that an ACM program has the completely-distributed/partially-related data references.

We uses the next equations as $E_{feild}(v(s))$ and $f_{field}(v(s))$ at each time-step shown in [95][98][99][115][116]:

$$E_{field}(v(s)) = -|\nabla I(v(s))|^2, \tag{5.10}$$

$$f_{field}(v(s)) = -\frac{1}{2}\frac{\partial E_{field}(v(s))}{\partial v(s)} = \nabla|\nabla I(v(s))|, \tag{5.11}$$

where $I(v(s))$ is a pixel value at the position of a partial contour $v(s)$ on a source image and $\nabla = (\partial/\partial x, \partial/\partial y)^T$. Equations5.10 and 5.11 define $E_{feild}(v(s))$ and $f_{field}(v(s))$ with only the coordinate values of $v(s)$, respectively. Equation5.10 means that the larger the gradient of a pixel value where a partial contour $v(s)$ stays is, the smaller $E_{field}(v(s))$ is. $f_{field}(v(s))$ pulls the partial contour $v(s)$ to the position of larger pixel-value gradient.

The external-constraint potential energy and the external constraint force

E_{con} and f_{con} are also defined using the coordinate values of at most the four neighboring contour points.

The external-constraint potential energy E_{con} is defined as the combining potential energy of the following two:

- E_{shape} : the shape potential energy
  - A potential energy causes the shape force f_{shape}.
  - The shape of an initial contour intended as the ROI shape is maintained by f_{shape}.
- $E_{distance}$: the distance potential energy
  - A potential energy generates the distance force $f_{distance}$.
  - $f_{distance}$ works on each contour point to make the distances between all the neighboring two contour points as even as possible.

Therefore, potential energy $E_{con}(v(s))$ and the external force that works $v(s)$ at each time-step are rewritten into the following equations:

$$E_{con}(v(s)) = E_{shape}(v(s)) + E_{distance}(v(s)), \tag{5.12}$$

$$f_{con}(v(s)) = f_{shape}(v(s)) + f_{distance}(v(s)). \tag{5.13}$$

– The shape potential energy and the shape force

E_{shape} causes a force f_{shape} that maintains the ROI shape given by an initial contour.

f_{shape} of each contour points is assumed as a spring force. E_{shape} and f_{shape} are defined as follows:

$$\begin{aligned} E_{shape}(v(s)) &= k_{shape}(s) \left| \left(\frac{v(s+1)-v(s)}{h(s)} - \frac{c(s+1)-c(s)}{h_i(s)} \right) \right. \\ &\quad \left. + \left(\frac{v(s-1)-v(s)}{h(s-1)} - \frac{c(s-1)-c(s)}{h_i(s-1)} \right) \right|^2 \\ &= k_{shape}(s) \ |d_{con-s}(v(s))|^2, \end{aligned} \tag{5.14}$$

$$\begin{aligned} f_{shape}(v(s)) &= k_{shape}(s) \left(\left(\frac{v(s+1)-v(s)}{h(s)} - \frac{c(s+1)-c(s)}{h_i(s)} \right) \right. \\ &\quad \left. + \left(\frac{v(s-1)-v(s)}{h(s-1)} - \frac{c(s-1)-c(s)}{h_i(s-1)} \right) \right) \\ &= k_{shape}(s) \ d_{con-s}(v(s)), \end{aligned} \tag{5.15}$$

where k_{shape} is a spring constant. $h(i)$ is a distance between $v(i)$ and $v(i+1)$: $h(i) = |v(i+1)-v(i)|$. $d_{con-s}(v(s))$ is a vector from the position of the contour point $v(s)$ to the basis:

$$\begin{aligned} d_{con-s}(v(s)) &= \left(\frac{v(s+1)-v(s)}{h(s)} - \frac{c(s+1)-c(s)}{h_i(s)} \right) \\ &\quad + \left(\frac{v(s-1)-v(s)}{h(s-1)} - \frac{c(s-1)-c(s)}{h_i(s-1)} \right). \end{aligned} \tag{5.16}$$

$c(s)$ represents a contour point of an initial contour. The suffix i denotes that it corresponds to the initial contour. Figure 5.5 illustrates $d_{con-s}(v(s))$. In Figure 5.5, each d is a unit vector:

$$\begin{aligned} d_{ap}(s) &= \frac{v(s-1)-v(s)}{h(s-1)}, \\ d_{an}(s) &= \frac{v(s+1)-v(s)}{h(s)}, \\ d_{ip}(s) &= \frac{c(s-1)-c(s)}{h_i(s-1)}, \\ d_{in}(s) &= \frac{c(s+1)-c(s)}{h_i(s)}, \end{aligned} \tag{5.17}$$

where suffixes p and n mean "to the previous contour point" and "to the next contour point", respectively.

Vector $b_{shape}(s)$ from the position of the contour point $v(s)$ to the basis is obtained as follows:

$$\begin{aligned} b_{shape}(s) &= d_{con-s}(v(s)) + v(s) \\ &= \left(\left(\frac{v(s+1)-v(s)}{h(s)} - \frac{c(s+1)-c(s)}{h_i(s)}\right)\right. \\ &\quad \left. + \left(\frac{v(s-1)-v(s)}{h(s)} - \frac{c(s-1)-c(s)}{h_i(s)}\right)\right) + v(s). \end{aligned} \tag{5.18}$$

– The distance potential energy and the distance force

$E_{distance}(v(s))$ generates the distance force $f_{distance}(v(s))$ that makes the distances from the contour point $v(s)$ to two neighboring contour points as even as possible. $E_{distance}$ is defined as follows:

$$\begin{aligned} E_{distance}(v(s)) &= k_{distance}(s) \mid h_{average}(s) - |v(s+1)-v(s)| \mid^2 \\ &= k_{distance}(s) \mid h_{average}(s) - h(s) \mid^2 \end{aligned} \tag{5.19}$$

where $k_{distance}$ is a spring constant. $h_{average}(s)$ is the mean distance of $h(s)$ and $h(s-1)$:

$$\begin{aligned} h_{average}(s) &= \frac{|v(s+1)-v(s)| + |v(s)-v(s-1)|}{2} \\ &= \frac{h(s)+h(s-1)}{2}. \end{aligned} \tag{5.20}$$

By adopting Eq.5.20 to Eq.5.19, $E_{distance}(v(s))$ is rewritten as follows:

$$\begin{aligned} E_{distance}(v(s)) &= k_{distance}(s) \left(\frac{|v(s+1)-v(s)| - |v(s)-v(s-1)|}{2}\right)^2 \\ &= k_{distance}(s) \left(\frac{h(s)-h(s-1)}{2}\right)^2. \end{aligned} \tag{5.21}$$

On the other hand, $f_{distance}$ is defined as follows:

$$f_{distance}(v(s)) = k_{distance}(s)\ (b_{distance}(v(s)) - v(s)) \tag{5.22}$$

where $b_{distance}(v(s))$ is the position vector of the basis point for $f_{distance}(v(s))$. $b_{distance}(v(s))$ is given by $|b_{distance}(v(s)) - v(s-1)| = |b_{distance}(v(s)) - v(s+1)|$. It is comparatively complicated to calculate the $f_{distance}$ due to the difficulty of implementing $b_{distance}$ while it is simple to execute $E_{distance}$ with Eq.5.21.

In this thesis, the attributes of an ellipse are utilized to obtain $b_{distance}$. $b_{distance}$ is indicates as follows:

$$\begin{aligned} & b_{distance}(v(s)) \\ = \ & \frac{v(s+1)+v(s-1)}{2} \\ & \pm\frac{n}{2}\sqrt{\left(|v(s+1)-v(s)| + |v(s)-v(s-1)|\right)^2 - |v(s+1)-v(s-1)|^2} \end{aligned} \tag{5.23}$$

where n is a normal vector on the perpendicular bisector of the line segment from $v(s-1)$ to $v(s+1)$. Concerning the sign $\pm$ in Eq.5.23, the equation chooses the sign that makes $|b_{distance}(v(s)) - v(s)|$ larger. Appendix B at the end of this thesis gives the concrete derivation of $b_{distance}(v(s))$.

Accordingly, $f_{distance}(v(s))$ is redefined as follows:

$$\begin{aligned} & f_{distance}(v(s)) \\ = \ & k_{distance}(s)\ (b_{distance}(v(s)) - v(s)) \\ = \ & k_{distance}(s)\ \left(\frac{v(s+1)+v(s-1)}{2}\right. \\ & \left.\pm\frac{n}{2}\sqrt{\left(|v(s+1)-v(s)| + |v(s)-v(s-1)|\right)^2 - |v(s+1)-v(s-1)|^2}\right) \end{aligned}$$

$$
\begin{aligned}
& -v(s) \Big) \\
= \; & k_{distance}(s) \left(\frac{v(s+1)+v(s-1)-2v(s)}{2} \right. \\
& \left. \pm \frac{n}{2} \sqrt{\left(|v(s+1)-v(s)|+|v(s)-v(s-1)|\right)^2 - |v(s+1)-v(s-1)|^2} \right) \\
= \; & k_{distance}(s) \left(\frac{(v(s+1)-v(s))+(v(s-1)-v(s))}{2} \right. \\
& \left. \pm \frac{n}{2} \sqrt{\left(|v(s+1)-v(s)|+|v(s)-v(s-1)|\right)^2 - |v(s+1)-v(s-1)|^2} \right) \\
= \; & k_{distance}(s) \;\; d_{con-d}(v(s)) \qquad (5.24)
\end{aligned}
$$

where $d_{con-d}(v(s)) = \frac{(v(s+1)-v(s))+(v(s-1)-v(s))}{2}$
$\pm \frac{n}{2} \sqrt{\left(|v(s+1)-v(s)|+|v(s)-v(s-1)|\right)^2 - |v(s+1)-v(s-1)|^2}$.

Figure 5.5 also shows d_{con-d}.

Hence, using $b_{distance}$ or d_{con-d}, $E_{distance}$ is rewritten as follows:

$$
\begin{aligned}
E_{distance}(v(s)) &= k_{distance}(s) \left(\frac{|v(s+1)-v(s)|-|v(s)-v(s-1)|}{2} \right)^2 \\
&= k_{distance}(s) \left(\frac{h(s)-h(s-1)}{2} \right)^2 \\
&= k_{distance}(s) \; |b_{distance}(v(s))-v(s)|^2 \\
&= k_{distance}(s) \; |d_{con-d}(v(s))|^2. \qquad (5.25)
\end{aligned}
$$

As a result, $E_{con}(v(s))$ and $f_{con}(v(s))$ are redefined as follows:

$$
\begin{aligned}
E_{con}(v(s)) &= E_{shape}(v(s)) + E_{distance}(v(s)) \\
&= k_{shape}(s) \left| \left(\frac{v(s+1)-v(s)}{h(s)} - \frac{c(s+1)-c(s)}{h_i(s)} \right) \right. \\
&\qquad \left. + \left(\frac{v(s-1)-v(s)}{h(s)} - \frac{c(s-1)-c(s)}{h_i(s)} \right) \right|^2 \\
&\quad + k_{distance}(s) \; |b_{distance}(v(s))-v(s)|^2
\end{aligned}
$$

$$= k_{shape}(s)\ |d_{con-s}(v(s))|^2 + k_{distance}(s)\ |d_{con-d}(v(s))|^2, \quad (5.26)$$

$$\begin{aligned} f_{con}(v(s)) &= f_{shape}(v(s)) + f_{distance}(v(s)) \\ &= k_{shape}(s) \left(\left(\frac{v(s+1)-v(s)}{h(s)} - \frac{c(s+1)-c(s)}{h_i(s)} \right) \right. \\ &\qquad \left. + \left(\frac{v(s-1)-v(s)}{h(s)} - \frac{c(s-1)-c(s)}{h_i(s)} \right) \right) \\ &\quad + k_{distance}(s)\ (b_{distance}(v(s)) - v(s)) \\ &= k_{shape}(s)\ d_{con-s}(v(s)) + k_{distance}(s)\ d_{con-d}(v(s)). \end{aligned} \quad (5.27)$$

The discretized equation-of-motion of an active contour in ACM with consideration of an ROI shape

From Eq.5.2, 5.10 and 5.26, $E_{snake}(v(s))$ at $v(s)$ at each time-step is given as follows:

$$\begin{aligned} E_{snake}(v(s)) &= E_{int}(v(s)) + E_{field}(v(s)) + E_{con}(v(s)) \\ &= \left(\omega_1(s) \left| \frac{v(s+1)-v(s)}{h(s)} \right|^2 + \omega_2(s) \left| \frac{v(s+1)-2v(s)+v(s-1)}{(\frac{h(s)+h(s-1)}{2})^2} \right|^2 \right) \\ &\quad - |\nabla I(v(s))|^2 \\ &\quad + \left(k_{shape}(s)\ |d_{con-s}(v(s))|^2 + k_{distance}(s)\ |d_{con-d}(v(s))|^2 \right) \end{aligned} \quad (5.28)$$

where the backward difference formulae and the central difference formulae of $v(s)$ by s are used for E_{int}.

Besides, from Eqs.5.6, 5.8, 5.11 and 5.27, the discretized equation-of-motion of $v(s)$ in an ACM with consideration of an ROI shape is obtained as follows:

$$\mu v_{tt}(s) + \gamma v_t(s)$$
$$+ \left(a_{(s-2)} v(s-2) + b_{(s-1)} v(s-1) + c_s v(s) + b_s v(s+1) + a_s v(s+2) \right)$$

$$
\begin{aligned}
\Big(= \quad & f_{field}(v(s)) + f_{con}(v(s))\Big) \\
\Big(= \quad & f_{field}(v(s)) + (f_{shape}(v(s)) + f_{distance}(v(s))) \Big) \\
= \quad & \nabla|\nabla I(v(s))| \\
& + \Big(k_{shape}(s)\ d_{con-s}(v(s)) + k_{distance}(s)\ d_{con-d}(v(s)) \Big). \qquad (5.29)
\end{aligned}
$$

Equation5.29 indicates that the equation-of-motion of each contour point $v(s)$ needs the coordinate values of itself and only four neighboring points. Moreover, the coordinate values of itself and only two neighboring points are used to calculate the potential energy of a contour point $E_{snake}(v(s))$ as described in Eq.5.28. In parallel processing of the ACM program, the execution of the equation-of-motion and $E_{snake}(v(s))$ of each contour point $v(s)$ is treated as a task. Accordingly, each task has the dependency with four other tasks.

Analysis of the task allocation as the selection of proper data-reference locality

The dependency of tasks, which implies the data-reference locality, must be considered when multiple tasks are allocated into processing elements. As examined in the previous section, after preferentially executing the tasks whose results are to be passed, each processing element may execute the remaining tasks and communicate with other processing elements simultaneously. This simultaneous treatment of task execution and communication is effective for parallel processing of the ACM program with such a task dependency. It must, however, be noted that the allocation of multiple tasks gives rise to a trade-off between the parallel-processing efficiency with hiding the communication latency and high-speed parallel processing of the program. The more the number of tasks allocated to a processing element is, the more easily the communication latency can be hidden. On

the other hand, the execution time of tasks increases in a processing element.

The tradeoff is also affected by the system ability such as task-execution power of processing elements and communication throughput among processing elements. At least three factors as parameters of the system ability must be considered; i.e., the communication time for the unit data between two processing elements, set-up time for one communication transaction and execution time of a task in a processing element.

Consider the following two examples assuming the transactions that processing element A executes two tasks and passes the two data as the results to another processing element B. Let the unit data be one data needed for the execution of a task for simplicity. T_{exec}, T_{comm} and T_{setup} represent the execution time of a task, the communication time of one data and set-up time for the communication transaction, respectively.

Case 1 $T_{exec} \ll T_{comm}, T_{setup}$

1. A executes the first task.
2. After the execution of the first task taking T_{exec}, A begins set up the communication.
3. A finished the execution of the second task taking T_{setup} during setting up the communication.
4. At last, A can successively pass the two data after the setting up.

As a result, the total time of these four transactions $T_{1\ total}$ is $T_{1\ total} = T_{exec} + T_{setup} + 2 \times T_{comm}$. In this case, the communication time is dominant in the total time and the utilization ratio of A is low. As regards parallel processing of the

program, this means that the high parallel-processing efficiency cannot be achieved.

Case 2 $T_{exec} \gg T_{comm} + T_{setup}$

1. A executes the first task.
2. After the execution of the first task taking T_{exec}, A begins set up the communication.
3. A finished the communication of the first data in $T_{setup} + T_{comm}$ during the execution of the second task.
4. A communicates the second data taking $T_{setup} + T_{comm}$ after executing the second task because the setting up for the communication of the second data is required again.

Consequently, the total time of these transactions is $T_{2\ total} = 2 \times T_{exec} + T_{setup} + T_{comm}$. In this case, the execution time is dominant in the total time and the utilization ratio of A is high. Concerning the parallel processing, this indicates that the high parallel-processing efficiency can be obtained.

Nevertheless, it is doubtful where $T_{2\ total}$ is greater than $T_{1\ toral}$ because these total time are dependent on the system parameters; i.e., T_{exec}, T_{setup} and T_{comm}. Therefore, it would be indispensable to select the appropriate number of tasks, which implied data-reference locality, treated in a processing element with considering these system abilities.

In Eq. 5.1, the potential energy E_{snake} of an active contour is calculated by summing up all the potential energies of contour points, each of which is represented by $E_{snake}(v(s))$. It takes much shorter time to sum up $E_{snake}(v(s))$ than the one to execute a task including

the equation-of-motion and $E_{snake}(v(s))$ of a contour point. Summing up of $E_{snake}(v(s))$ every time may still decrease the parallel processing efficiency of the ACM program due to increasing communications among processing elements.

Theoretically, when the active contour fits into an ROI contour, each $E_{snake}(v(s))$ falls into a constant or the steady state with slight vibrations, which means the convergence of $E_{snake}(v(s))$. The convergence of $E_{snake}(v(s))$ can be decided by the processing element that executes $E_{snake}(v(s))$. In terms of the proposed ACM and image processing system, at each time-step in parallel processing of the program, a processing element does not communicate $E_{snake}(v(s))$ with other processing elements but holds $E_{snake}(v(s))$ as a history. Each processing element calculates the variance (or standard deviation) of $E_{snake}(v(s))$ in the history, and decides its convergence if the variance is less than a threshold ε.

5.3.2 Performance evaluation

A pair of the equation-of-motion and the potential energy $E_{snake}(v(s))$ of a contour point $v(s)$ is executed as a task in each processing element with Eqs. 5.8, 5.28 and 5.29. The task-level parallel processing of the ACM program is realized by the proposed ACM. This chapter has mentioned that the image processing system based on the distributed-memory parallel computer architecture is effective for the parallel processing of the ACM program. This system has an interconnection network that enables processing elements to communicate locally with each other. Besides, when multiple tasks are allocated to a processing element, it is valuable for a processing element to execute tasks and communicate with other processing elements simultaneously with preferentially executing the tasks whose results are to be passed. The efficient parallel processing of the ACM program could be

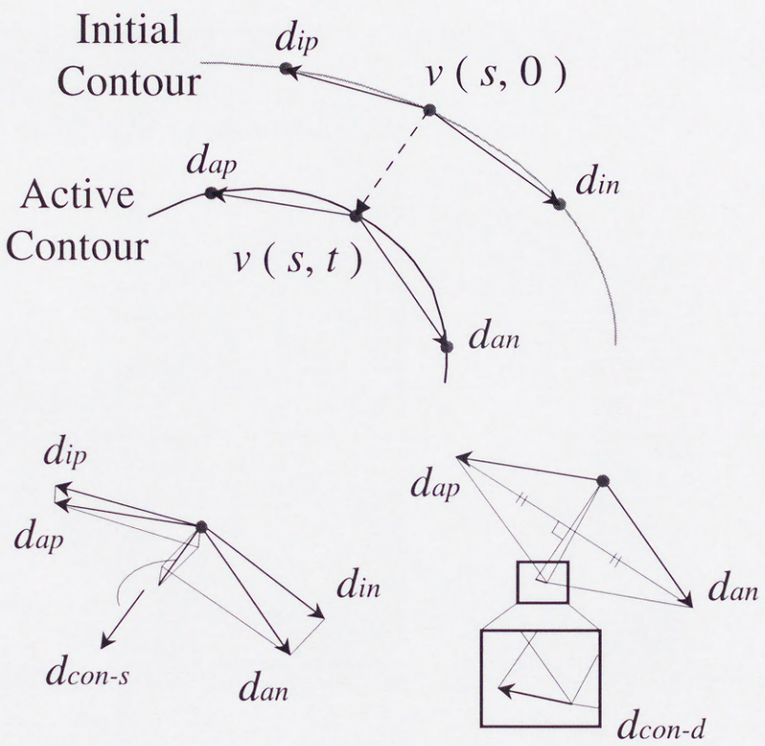


Figure 5.5: Constraint forces.

achieved with the system and parallel processing technique.

The allocation of multiple tasks to a processing element, however, causes a tradeoff between the parallel-processing efficiency with hiding the communication latency and high-speed parallel processing of the program. The tradeoff is also affected by the system abilities; e.g., task-execution power of processing elements and communication throughput among processing elements. Therefore, it would be indispensable to select the appropriate number of tasks, which implies data-reference locality, treated in a processing element with considering these system abilities.

This chapter discusses the parallel processing efficiency of the ACM program with the image processing system presented in Section 5.2.2 in terms of the selection of proper data-reference locality in the ACM program. Software simulations of the parallel processing have been carried out while extracting a region-of-interest (ROI) from the source images with the ACM.

Parameters for experiments

According to the analysis in Section 5.3.1, let parameters consist of the three factors; i.e., the communication time between two processing elements for passing coordinate values of a contour point, set-up time for one communication transaction and the execution time of a task in a processing element. Table 5.1 indicates these processing times. The unit of the communication between processing elements is the coordinate value of a contour point. Data size of the coordinate values as a communicated unit is 64 bits $\times$ 2 and the communication time is $T_{comm} = 6.4[\mu sec]$ derived from 100 [Mbits/sec] $\times$ 20%. The data

size is really used in the ACM program. The throughput of a network is almost equal to that of the 100 Mbits ethernet broadly used in recent years. The 100 Mbits ethernet allows for a raw data transfer rate of 100 megabits per second, with actual throughputs in the range of 2 to 3 megabits per second; i.e. 20% to 30%. Let the set-up time 1.28[μsec]. The set-up time is defined relatively long for evaluating the parallel-processing performance of the system. In the experiments, the execution time of a task in a processing element is determined by using these two parameters. Three of the task execution times are attached to Table 5.1 for example. Table 5.1 also shows the timing ratios under making communication time 1.0 as the basis to compare with each other. In addition, timing ratios of the execution time to communication time varies 0.1, 0.6 and 1.2. The first ratio (0.1) means that the task-processing ability of a processing element is extremely higher than the communication throughput. The second one (0.6) and the third one (1.2) represent the relatively higher and lower ability of a processing element, respectively.

In the experiments regarding the parallel processing efficiency of the ACM program, the ROI extraction was actually carried out. Figures 5.6 and 5.7 show examples of the results in ROI extraction with the proposed ACM.

Figure 5.6 shows the source image that is an artificial image with Gaussian noise. The ROI has a large concavity. Figures 5.6 (a) and (b) indicate the results of the ROI extraction without and with the external constraint forces, respectively. An active contour was given as an initial contour that was the most outer contour, and then works on the source image for 6877 time-steps. (a) represents that the active contour could not fit the concave part of the ROI by the ACM without the external forces. By contrast, the active contour could fit the concave part of the ROI in the ACM with the external forces in (b).

These experimental results in Figure 5.6 mean the reliability of the proposed ACM as a image processing technique.

Figure 5.7 is a result of the ROI extraction on MR (magnetic resonance) image. The ROI is a cerebellum of a human head. Figure 5.7 (a) is the source image with the initial contour of a white curve. (b) represents gradients $|\nabla I|$ of pixel values in the region around and within the initial contour line. The lighter a pixel of the image is, the larger the gradient is. The initial contour is also illustrates in (b). (c) depicts the result of the cerebellum extraction from the MR image. (c) also includes the initial contour and the last position of the active contour, which stands for the cerebellum contour. (d) illustrates the traveling of the active contour from the initial position to the last position in 7105 time-steps. The experimental result in Figure 5.7 shows the applicability of the proposed ACM.

The experiments are performed to measure the execution time of the ACM program. The number of contour points, that is, tasks is 32 in the experiments. An active contour is assumed to work for the 10000 time-steps because the time-steps needed for an active contour converging depends on many parameters; e.g., initial location of the active contour and the weights given to the forces. Timing ratios of the task execution time T_{exec} to the communication time T_{comm} varies from 0.1 to 3.2. The numbers of processing elements are 1, 2, 4, 8, 16, and 32. The other parameters are the same as those for the experiment of Figure 5.6.

Table 5.1: Parameters for the experiments of parallel processing.

Parameters	Processing time [μ sec]	Timing ratio
Communication	6.40	1
Set-up time for communication	1.28	0.20
Equation-of-motion Execution	0.64	0.10
	3.84	0.60
	7.68	1.20

Experimental results

Figure 5.8 shows the experimental results. In this figure, "Execution" and "Communication" stand for the cases that the execution time T_{exec} and communication time T_{comm} are dominant in the total time, respectively. This figure indicates that, when the task-processing ability of a processing element is higher ($T_{exec}/T_{comm} = 0.6$) and the number of processing elements is comparatively small, the high-speed processing and high parallel-processing efficiency of the ACM program can be achieved. In this case, if the number of processing elements is large, the parallel-processing efficiency is decreased and the execution time of the program is sometime increased (e.g., 32) because the communication latency becomes a bottleneck of the parallel processing. By contrast, the high-speed processing and the high parallel-processing efficiency can be achieved according to the number of processing element when the task-processing ability of a processing element is lower and/or the communication throughput is higher ($T_{exec}/T_{comm} = 1.2$) with large

number of processing elements. If the task-execution ability of a processing element is extremely high ($T_{exec}/T_{comm} = 0.1$), it seems that the parallel processing is not always needed. This was caused in the case where the number of tasks was small of 32 in the experiments. The task-level parallel processing of the ACM program must be effective and essential in the case of the more practical use such the case where the proposed ACM is applied to the medical images with large number of contour points.

Assume that the parallel-processing efficiency of the ACM program is valuable only in the following cases:

- in terms of high-speed parallel processing

  The processing time of the ACM program is under 1.0 [sec].

- in terms of parallel-processing efficiency

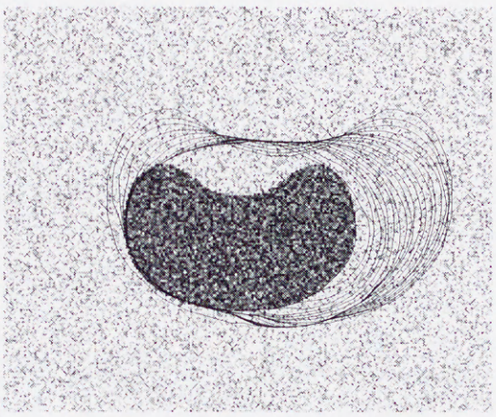
  The execution time of tasks is dominant in the total time.

Figure 5.8 describes that when the timing ratios of the execution and the communication are 0.6 and 1.2, the valuable numbers of processing elements are 2 and both 4 and 8, respectively. This means that the efficient parallelism of tasks varies according to the ratio of execution/communication.

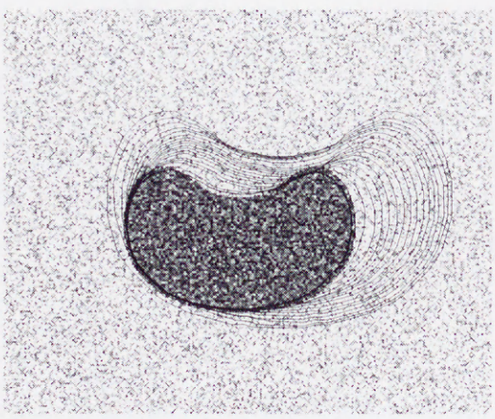
Figure 5.9 represents the processing time of the ACM program when the number of processing elements and the timing ratio of the task execution to the data communication vary from 1 to 32 and from 0.1 to 3.2, respectively. In Figure 5.9, the processing times of the program are truncated if over 1.1 [sec]. Figure 5.9 also means the efficient parallelism of tasks.

With respect to the high-speed parallel processing of the ACM program and high parallel-processing efficiency, Figures 5.8 and 5.9 indicate the significance of selecting the appropriate number of tasks treated in a processing element. The proper number of tasks is also affected by several system abilities such as task-execution power of processing elements and communication throughput among processing elements. These experimental results are applicable to the parallel processing of program based on the ACM that allows users to interactively operate contour points and enables contour points to increase or decrease.

Selecting the appropriate number of tasks means exploiting proper data-reference locality of tasks. As a consequence, for the parallel processing of an SPMD program with completely-distributed/partially-related data references, the effectiveness of the distributed-memory parallel computer architecture with an interconnection network and the indispensability of selecting proper data-reference locality are confirmed through these experiments.



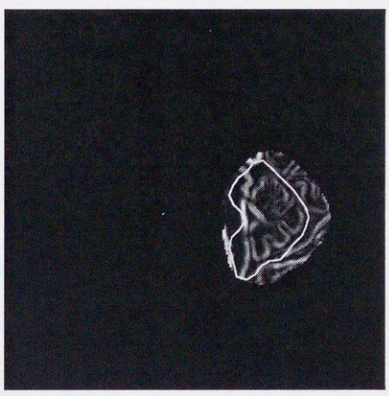
(a) Without the external forces.



(b) With the external forces.

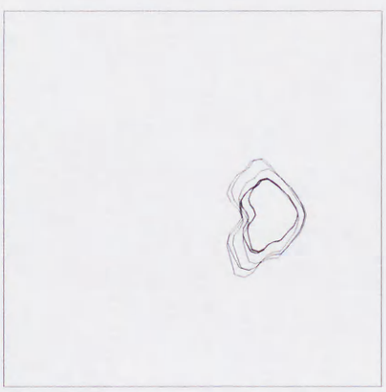
Figure 5.6: Examples of experimental results with/without the external forces in the case of an artificial image.

(a) A source MR image of a human head.



(b) Gradient of pixel values in the source image.

(c) A result of the cerebellum extraction.



(d) Motion of the active contour on the source image.

Figure 5.7: A result of region-of-interest extraction with a medical image.

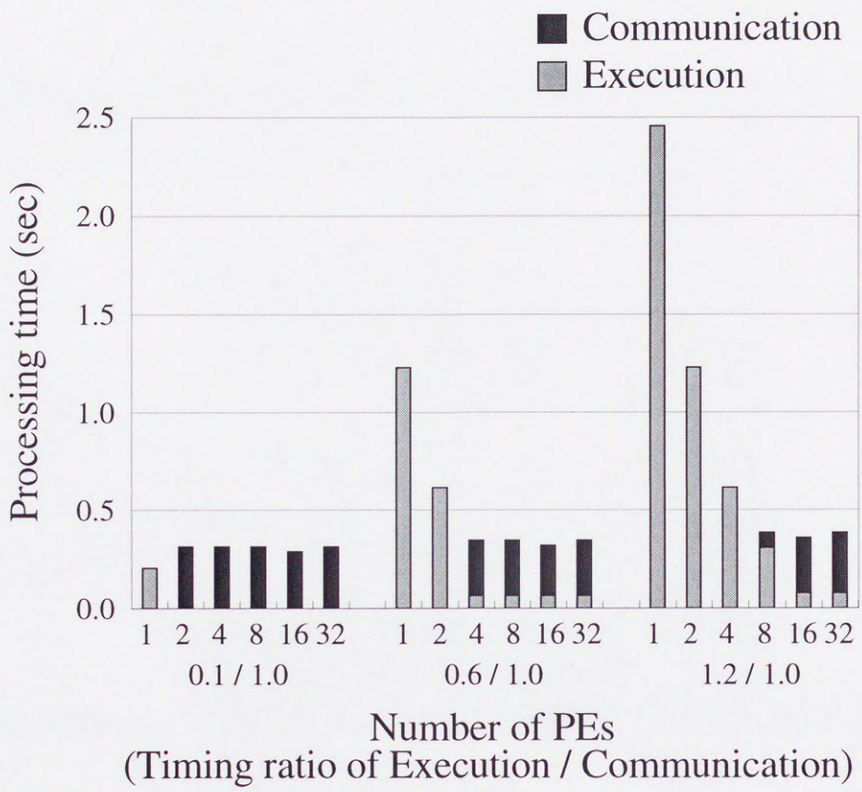


Figure 5.8: Execution time of the ACM program (timing ratios are 0.1, 0.6, 1.2.).

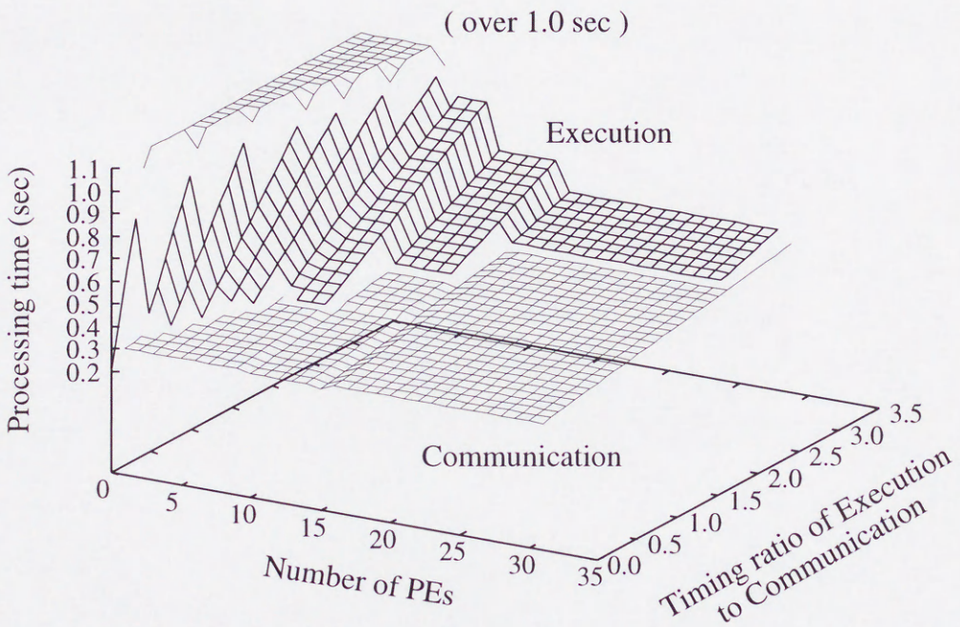


Figure 5.9: Execution time of the ACM program (timing ratio varies from 0.1 to 3.2.).

5.4 Conclusions

In Chapter 5, respecting a program of an active contour model (ACM) as an SPMD program with completely-distributed/partially-related data references, parallelism of the program and its parallel computer architecture has been examined at the beginning.

Next, an active contour model with considering local shapes of ROI (region-of-interest) has been proposed to solve the difficulty in parallel processing with respect to programs based on conventional models. The proposed model can enhance the parallel processing efficiency of the program by utilizing the partial relations of data reference effectively.

This chapter also has experimented on the parallel processing efficiency of an image-processing program based on the proposed model. This experiments has been carried out on a distributed-memory parallel processing system with an interconnection network, in which each processing element can simultaneously treat the execution of tasks and communication with the others.

The experimental results have confirmed that the system and the proposed model are valuable to parallel processing of ACM programs. This implies that the distributed-memory parallel computer architecture with an interconnection network and the selection of proper data-reference locality are essential for the parallel processing of an SPMD program with completely-distributed/partially-related data references.

6 Conclusions

Recent application programs become larger and more complicated with huger data. The increasing requirement of high-speed parallel processing with practical application programs is the motive for this project. This project has focused on the SPMD (single-program multiple-data) programming model, and investigated data-reference locality of programs based on the model with practical application programs.

We have proposed a classification of SPMD programs into the three types of data reference at the beginning, and discussed parallel computer architectures suitable for the types of programs then. Moreover, several experiments on the SPMD application programs and parallel processing systems have also been carried out to clarify the relationship among SPMD programs and parallel computer architectures. Here, this study is summarized as follows:

Chapter 2 has provided a classification of SPMD programs. SPMD programs have been classified into three types of data reference to parallel computer architectures. The data-reference types are as follows: 1) the type of a locality-changeable data reference, 2) the type of a completely-distributed / completely-related data reference, and 3) the type of a completely-distributed / partially-related data reference. After that, this chapter has examined parallel computer architectures for the data-reference types of programs.

Consequently, the relationship among the data-reference types of SPMD programs and parallel computer architectures are represented as follows:

1) the locality-changeable type and
the distributed-memory parallel computer architecture with a shared memory

2) the completely-distributed / completely-related type and
the distributed-memory parallel computer architecture with an interconnection network

3) the completely-distributed / partially-related type and
the distributed-memory parallel computer architecture with an interconnection network

Chapter 3 has referred to a functional program as an SPMD program with a locality-changeable data reference, and proposed a hierarchical parallel reduction system for the program. This system is based on a distributed-memory parallel processing system with a shared memory for active use of the data-reference locality. Through the analysis of static task scheduling strategies, this chapter has also proposed a dynamic task scheduling strategy for the program, which is called the breadth-first/depth-first task scheduling strategy. Then, the effectiveness of the parallel processing system and task scheduling strategy proposed in this chapter has been indicated by several experiments. Th means that the size of task sets have to be changed according to utilization ratio of processing elements and parallelism of tasks. Therefore, as regards an SPMD program with a locality-changeable data reference, the distributed-memory parallel computer architecture with a shared memory is appropriate, and a task scheduling strategy to change the size of task

sets dynamically is essential.

Chapter 4 has treated data-parallel volume rendering as an SPMD program with a completely-distributed / completely-related data reference, and described an image construction system, which is a distributed-memory parallel processing system with an interconnection network to achieve high-speed processing of volume rendering. After that, this chapter has applied the binary-swap method to volume rendering algorithm, and proposed adaptive volume subdivision so that the processing time of data communication can be reduced with the load balance of processing elements maintained. This is generally because the communication time for data scattering and gathering in the SPMD program markedly augments as the number of processing elements increases. Moreover, this chapter has argued about the parallel processing system and the proposed techniques, and carried out some experiments on them. The experimental results have shown the effectiveness of a distributed-memory parallel processing system with an interconnection network and techniques to decrease the processing time of data communication for the SPMD program. It consequently means that making efficient the data communication among processing elements is required rather than achieving high-speed processing of parallel tasks for an SPMD program with a completely-distributed / completely-related data reference.

In chapter 5, respecting a program of an active contour model (ACM) as an SPMD program with a completely-distributed / partially-related data reference, parallelism of the program and a parallel computer architecture for it have been examined at the beginning. An active contour model with considering local shapes of ROI (region-of-interest) has been proposed next against the difficulty in parallel processing of programs specified by conventional models. The proposed model can enhance the parallel processing efficiency of the

program by utilizing the partial relations of data references effectively. After that, this chapter has experimented on parallel processing of an image-processing program based on the proposed model with a distributed-memory parallel processing system including an interconnection network. It has been confirmed through the experimental results that the system and the proposed model are valuable to parallel processing of ACM programs. This implies that the distributed-memory parallel computer architecture with an interconnection network and the selection of proper data-reference locality are essential for the parallel processing of an SPMD program with a completely-distributed / partially-related data reference.

The research in each chapter has clarified the relationship between each data-reference type of SPMD programs and parallel computer architecture. Table 6.1 describes the relationship as a summary with techniques for efficient parallel processing of the program. Therefore, we have indicated a policy for parallel processing that is the largest and still increasing requirement of practical application programs.

There are some attractive topics to be investigated around this project. Concerning an SPMD program with a locality-changeable data reference, the irregularity of task processing time leads to the interest in speculative generation of tasks[117][118]. High-speed parallel processing of the SPMD programs could be achieved with the speculative generation increasing task parallelism. Regarding SPMD programs with a completely-distributed / completely-related data reference and a completely-distributed / partially-related data reference, although parallel tasks generated from the SPMD programs are homogeneous, processing times of parallel tasks are not always even. Accordingly, higher parallel-processing efficiency of the SPMD programs would be acquired by active selection

Table 6.1: The data-reference types of SPMD programs and their suitable parallel computer architectures (parallel processing techniques).

The Data-Reference Type of an SPMD Program	Parallel Computer Architecture [Parallel Processing Technique]
locality-changeable	the distributed-memory with a shared memory [dynamic change of task-set size]
completely-distributed / completely-related	the distributed-memory with an interconnection network [reduction of passed data, improving the transaction of data scatter and gather]
completely-distributed / partially-related	the distributed-memory with an interconnection network [selecting proper locality of data reference]

of task sets, which include variable numbers of tasks, during the execution of the programs.

These topics are addressed as related researches in the future work.

Appendix A

Some primitive functions and combining forms of FL.

f : x	denotes a function f is applied to an object x;
f ∘ g	"∘" denotes the application operation, and it is expressed as "@" in reduction graphs;
<x1, x2, ..., xn>	denotes a sequence construction;
p→q; r	abbreviates cond : <p, q, r>.

```
id : x = x
si : <x1, ....xn> = xi
signal : x = exception
distl : <x, <y1,..., yn>> = <<x, y1>, ..., <x, yn>>
distr : <<x1, ..., xn>, y> = <<x1, y>, ..., <xn, y>>
trans : <x1, ..., xn> = <y1, ..., yn>        ; yj is the sequence of jth elements of xi 's
neg : x = -x                                 ; x is a number
tt : x = T
isnum : x = T                                ; if x is a number
isint : x = T                                ; if x is an integer
f ∘ g : x = f : ( g : x )
[f, g] : x = <f:x,  g:x>
α: f : <x1, ..., xn> = <f : x1, ..., f : xn>
seqof : p : <x1, ..., xn> = T                ; iff  p: xi = T  for all i = 1, ..., n
pcons : <p1, ..., pn> : <x1, ..., xn> = T    ; iff  pi : xi = T  for all  i = 1, ..., n
```

Appendix B

Derivation of $b_{distance}$

We utilize the attributes of an ellipse to obtain $b_{distance}$. Assume an ellipse constructed by $v(s)$ given with the constant $|v(s+1) - v(s)| + |v(s) - v(s-1)|$; i.e., two contour points $v(s+1)$ and $v(s-1)$ as focuses.

Let r_a and r_b be a major radius and a minor one of the ellipse, respectively. In terms of the radii and the contour points, attributes of an ellipse derive following equations:

$$2\sqrt{r_a^2 - r_b^2} = |\ v(s+1) - v(s-1)\ |, \tag{B.1}$$

$$r_a = \frac{|v(s+1) - v(s)| + |v(s) - v(s-1)|}{2}. \tag{B.2}$$

Equations B.1 and B.2 lead to r_b as follows:

$$r_b = \pm\frac{1}{2}\sqrt{(|v(s+1) - v(s)| + |v(s) - v(s-1)|)^2 - |v(s+1) - v(s-1)|^2}. \tag{B.3}$$

Since r_b is a minor radius of the ellipse, $b_{distance}(v(s))$ is defined as follows:

$$\begin{aligned} b_{distance}(s) &= \frac{v(s+1) + v(s-1)}{2} + r_b n \\ &= \frac{v(s+1) + v(s-1)}{2} \\ &\quad \pm\frac{n}{2}\sqrt{(|v(s+1) - v(s)| + |v(s) - v(s-1)|)^2 - |v(s+1) - v(s-1)|^2} \end{aligned} \tag{B.4}$$

where n is a normal vector on the perpendicular bisector of line segment $v(s-1)$ to $v(s+1)$. Concerning $\pm$, a sign that makes $|b_{distance}(s) - v(s)|$ smaller is chosen.

Bibliography

[1] K. Hwang and Z. Xu. *Scalable Parallel Computing.* Technology, Architecture, Programming. WCB/McGraw-Hill, 1998.

[2] Howard Jay Siegel, Seth Abraham, William L. Bain, Kenneth E. Batcher, Thomas L. Casavant, Doug DeGroot, Jack B. Dennis, David C. Douglas, Tse-Yun Feng, James R. Goodman, Alan Huang, Harry F. Jordan, J. Robert Jump, Yale N. Patt, Alan J. Smith, James E. Smith, Lawrence Snyder, Harold S. Stone, Russ Tuck, and Benjamin W. Wah. Report of the Purdue workshop on grand challenges in computer architecture for the support of high performance computing. *Journal of Parallel and Distributed Computing*, Vol. 16, No. 3, pp. 199–211, November 1992.

[3] P. Patton. Multiprocessors: Architecture and applications. *Computer*, Vol. 18, No. 6, pp. 29–40, 1985.

[4] Alan H. Karp. Programming for parallelism. *Computer*, Vol. 20, No. 5, pp. 43–57, May 1987.

[5] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, Vol. C-21, No. 9, pp. 948–960, September 1972.

[6] S. Yalamanchili and J. K. Aggarwal. Reconfiguration strategies for parallel architectures. *Computer*, Vol. 18, No. 12, pp. 44–61, December 1985.

[7] Daniel Lenoski, James Laudon, Truman Joe, David Nakahira, Luis Stevens, Anoop Gupta, and John Hennessy. The DASH prototype: Logic overhead and performance. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 1, pp. 41–61, January 1993.

[8] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica S. Lam. The Stanford Dash multiprocessor. *Computer*, Vol. 25, No. 3, pp. 63–79, March 1992.

[9] D. Adams. Cray tsd system architecture overview manual. Research report, Cray Research, Inc., September 1993.

[10] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms.* Reading, Mass. : Addison-Wesley Pub. Co., 1975 printing, 470 p. CALL NUMBER: QA76.6 .A36, 1975.

[11] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, Vol. 33, No. 8, pp. 103–111, August 1990.

[12] K. Hwang, C. Wang, C.-L. Wang, and Z. Xu. Resource scaling effects on MPP performance: The STAP benchmark implications. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 5, pp. 509–527, May 1999.

[13] Hiroyuki Kitajima. Performance evaluation of a functional architecture. Master's thesis, Department of Computer and Mathematical Sciences, Graduate School of Information Sciences, Tohoku University, 2 1995.

[14] Mitsuhiro Nakaizumi, Hiroyuki Kitajima, Hong Shen, Hiroaki Kobayashi, and Tadao Nakamura. A memory system of the parallel reduction machine. Technical report, Electrical and Information Engineers, Japan, 8 1996. (Tohoku Gakuin University).

[15] Hong Shen, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura. A hierarchical parallel reduction system for the functional language fl. In *Proceedings of the High Performance Computing Conference '94(HPCC '94)*, pp. 270–279, 9 1994. (Singapore).

[16] Hong Shen, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura. Performance evaluation of a hierarchical parallel reduction system for fl. In Masato

Takeichi, editor, *Functional Programming II JSSST'94*, Lecture Note/Software 10, pp. 159–174. Kindaikagakusya Co. Ltd., 11 1994. (Kyoto).

[17] Hiroyuki Kitajima, Mitsuhiro Nakaizumi, Hong Shen, Hiroaki Kobayashi, and Tadao Nakamura. Task scheduling strategies and their locality evaluation of memory references on a parallel graph reduction system. *IPSJ Transactions*, Vol. 37, No. 11, pp. 2020–2029, 11 1996.

[18] Hiroyuki Kitajima, Hong Shen, Masayuki Katahira, Hiroaki Kobayashi, and Tadao Nakamura. Performance studies of the fl hierarchical parallel reduction system. Technical Report 1, IPSJ HPC, 6 1995. (SIG-HPC, IBM Japan Ltd., Hakozaki, Tokyo).

[19] Hong Shen, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura. Task scheduling with locality consideration for a clustered parallel fl reduction system. In *Proceedings of the First Aizu International Symposium on Parallel Algorithms/Architecture Synthesis(pAs '95)*, pp. 234–240, 3 1995. (Aizu University).

[20] Hiroyuki Kitajima, Hong Shen, Hiroaki Kobayashi, and Tadao Nakamura. Optimization of scheduling strategies for a pipelined reduction machine. Technical Report 941-1, the Japan Society of Mechanical Engineers, 3 1994. (Tohoku University).

[21] Kentaro Sano, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura. Data-parallel volume rendering with adaptive volume subdivision. *IEICE Transactions on Information and Systems*, Vol. E83-D, No. 1, pp. 80–89, 1 2000.

[22] Yoshinori Kimura, Kentaro Sano, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura. A study of virtual reality system with volume rendering. Technical report, Electrical and Information Engineers, Japan, 8 1998. (Tohoku University).

[23] Kentaro Sano, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura. Adaptive volume-subdivision for efficient data-parallel volume rendering. Technical Report 2, IPSJ HPC, 10 1998. (SIG-HPC, Tohoku University).

[24] Kentaro Sano, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura. Parallel processing of the shear-warp factorization with the binary-swap method on a distributed-memory multiprocessor system. In James Painter, Gordon Stoll, and Kwan-Liu Ma, editors, *IEEE Parallel Rendering Symposium*, pp. 87–94,118. IEEE, November 1997. ISBN 1-58113-010-4.

[25] Hiroyuki Kitajima, Yoshiyuki Kaeriyama, Hiroaki Kobayashi, and Tadao Nakamura. An active contour model using local shape information of a region-of-interest and its parallel processing. submitted to IPSJ Transactions on High Performance Computing Systems.

[26] Hiroyuki Kitajima, Yoshiyuki Kaeriyama, Hiroaki Kobayashi, and Tadao Nakamura. An active contour model using regions-of-interest shape information. Technical report, IPSJ, 9 1999. (Iwate Prefectural University).

[27] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *ACM Symposium on Theory of Computing (STOC '83)*, pp. 1–9, Baltimore, USA, April 1983. ACM Press.

[28] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and van Eicken Thorsten. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pp. 1–12, May 1993.

[29] Clyde P. Kruskal and Alan Weiss. Allocating independent subtasks on parallel processors. *IEEE Transactions on Software Engineering*, Vol. 11, No. 10, pp. 1001–1016, October 1985.

[30] Zhiwei Xu and Kai Hwang. Early prediction of MPP performance: The SP2, T3D, and Paragon experiences. *Parallel Computing*, Vol. 22, No. 7, pp. 917–942, October 1996.

[31] Z. Xu and K. Hwang. Mpp versus clusters for scalable computing. In *The 2nd International Symposium on Parallel Architectures, Algorithms, and Networks*, pp. 117–123. IEEE Computer Society Press, June 1996.

[32] V. Chaudhary and J. K. Aggarwal. A generalized scheme for mapping parallel algorithms. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 3, pp. 328–346, March 1993.

[33] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe. Task allocation in distributed data processing. *Computer*, Vol. 13, pp. 57–69, 1980.

[34] H. Kasahara and S. Narita. Practical multiprocessing scheduling algorithms for efficient parallel processing. *IEEE Transactions on Computers*, Vol. 33, No. 11, pp. 1023–1029, November 1984.

[35] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, Vol. 14, No. 2, pp. 141–154, February 1988.

[36] M. Norman and P. Thanisch. Models of machines and computations for mapping in multicomputers. *ACM Computer Surveys*, Vol. 25, No. 9, pp. 263–302, September 1993.

[37] Arvind, D. Culler, and G. Maa. Assessing the benefits of fine-grain parallelism in dataflow programs. In *Proceedings of Supercomputing '88*, pp. 69–69, November 1988.

[38] D. Kuck, A. Sameh, R. Cytrom, A. Veidenbaum, C. Polychronopoulos, G. Lee, T. McDaniel, B. Leasure, C. Bechman, J. Davies, and C. Kruskal. The effects of program restructuring, algorithm change, and architecture choice on program performance. In *Proceedings of 1984 International Conference on Parallel Processing*, August 1984.

[39] S. P. Amarasinghe and M. S. Lam. Communication optimization and code generation for distributed memory machines. *ACM SIGPLAN Notices*, Vol. 28, No. 6, pp. 126–138, June 1993.

[40] Christian Foisy and Emmanuel Chailloux. Caml flight: A portable spmd extension of ml for distributed memory multiprocessors. In A. P. Wim Bohm and John T. Feo, editors, *Proceedings of High Performance Functional Computing '95*, pp. 83–96, April 1995.

[41] Arvind Krishnamurthy and Katherine Yelick. Optimizing parallel spmd programs. In Keshav Pingali, Uptal Banerjee, David Gelernter, Alex Nicolau, and David Padua, editors, *Proceedings of the 7th International Workshop on Languages and Compilers for Parallel Computing*, No. 892 in Lecture Notes in Computer Science, pp. 331–345, Ithaca, New York, August 1994. Springer-Verlag.

[42] J. F. de Ronde, A. W. van Halderen, A. de Mes, M. Beemster, and P. M. A. Sloot. Automatic performance estimation of spmd programs on mpp. In W. Smit L. Dekker and J. C. Zuidervaart, editors, *Massively Parallel Applications and Development*, pp. 381–388, Delft, The Netherlands, June 1994. Elsevier, North-Holland.

[43] Herbert H. J. Hum and Guang R. Gao. Supporting a dynamic spmd model in a multithreaded architecture. In *Proceedings of the 1993 IEEE International Conference, COMPCON*, pp. 165–175, February 1993.

[44] A. F. Bashir, V. Susarla, and K. Vairavan. A statistical study of the performance of a task scheduling algorithm. *IEEE Transactions on Computers*, Vol. 32, No. 8, p. 774, August 1983.

[45] D. Bernstein, M. Rodeh, and I. Gertner. On the complexity of scheduling problems for parallel/pipelined machines. *IEEE Transactions on Computers*, Vol. 38, No. 9, pp. 1308–1313, September 1989.

[46] D.-K. Chen, H.-M. Su, and P.-C. Yew. The impact of synchronization and granularity on parallel systems. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 239–248, May 1990.

[47] S. Kortas and P. Angot. A practical and portable model of programming for iterative solvers on distributed memory machines. *Parallel computing*, Vol. 22, No. 4, pp. 487–512, June 1996.

[48] A. Singh and V. Van Dongen. An integrated performance analysis tool for spmd data-parallel programs. *Parallel computing*, Vol. 23, No. 8, pp. 1089–1112, August 1997.

[49] R. Cytron, J. Lipkis, and E. Schonberg. A compiler-assisted approach to spmd execution. In *Proceedings of Supercomputing '90*, pp. 398–406, New York, November 1990.

[50] John Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. In *ACM Turing Award Lectures: The First Twenty Years*, pp. 63–130. ACM Press, 1987.

[51] K. H. Haskell and R. J. Hanson. An algorithm for linear least squares problems with equality and nonnegativity constraints. *Mathematical Programming*, Vol. 21, pp. 98–118, 1981.

[52] David A. Turner. Miranda: A non-strict functional language with polymorphic types. In J. Jouannaud, editor, *Proceedings IFIP International Conference on Functional Programming Languages and Computer Architectures, Nancy, France*, Vol. 201 of *Lecture Notes in Computer Science*, pp. 1–16. Springer-Verlag, New York, NY, September 1985.

[53] J. Backus, J. H. Williams, and E. L. Wimmers. FL language manual (preliminary version). Research Report RJ 5339 (54809), IBM Almaden Research Center, San Jose, CA, 1986.

[54] John Backus, John H. Williams, Edward L. Wimmers, Peter Lucas, and Alexander Aiken. FL Language Manual, Parts 1 and 2. Research Report RJ 7100 (67163) 10/26/89, IBM Almaden Research Center, San Jose, California, October 1989.

[55] J. Backus, J. Williams, and E. Wimmers. An introduction to the functional language FL. In D. A. Turner, editor, *Research Topics in Functional Programming*, pp. 219–247. Addison-Welsey, Reading, MA, 1990.

[56] Lennart Augustsson and Thomas Johnsson. Parallel graph reduction with the $< \nu$, G $>$ -machine. In *Functional Programming and Computer Architecture*, pp. 202–214. ACM, 89.

[57] Lennart Augustsson and Thomas Johnsson. Lazy ML user's manual. Technical report, Department of Computer Science, Chalmers University of Technology, 1991.

[58] Lennart Augustsson and Thomas Johnsson. *Lazy ML user's manual*, July 1992. Draft.

[59] John Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, Vol. 21, No. 8, pp. 613–641, August 1978. Reproduced in "Selected Reprints on Dataflow and Reduction Architectures" ed. S. S. Thakkar, IEEE, 1987, pp. 215-243.

[60] C. P. Wadsworth. *Semantics and pragmatics of the lambda calculus.* Ph.D. thesis, Programming Research Group, Oxford University, September 1971. (have bits).

[61] H. Shen, H. Kobayashi, and T. Nakamura. Developing the lambda calculus for fl-oriented parallel reductions. In *Third International Conference for Young Computer Scientists*, pp. 649–650, 1993.

[62] R. L. Graham. *Bounds on Multiprocessing Timing Anomalies*, Vol. 17. SIAM J. of Applied Mathematics, March 1969.

[63] K. Langendoen. *Graph Reduction on Shared-Memory Multiprocessors.* Ph.D. thesis, University of Amsterdam, Amsterdam, NL, 1993.

[64] R. F. H. Hofman, K. G. Langendoen, and W. G. Vree. Scheduling consequences of keeping parents as home. In *Parallel and Distributed Systems*, pp. 580–588, Taiwan, National Tsing Hwa University, 1992.

[65] Paul Hudak and Lauren Smith. Para-Functional Programming: A Paradigm for Programming Multiprocessor Systems. In *Thirteenth Annual ACM Symposium on Principles of Programming Languages*, pp. 243–254, St. Petersburg Beach, Florida, January 15–16, 1986. ACM Press, New York.

[66] C. A. Ruggiero and J. Sargeant. Control of parallelism in the manchester dataflow machine. In *Third Functional Programming Languages and Computer Architecture, LNCS*, Vol. 274, pp. 1–15, 1987.

[67] Eric Mohr, David A. Kranz, and Robert H. Halstead, Jr. Lazy task creation: A technique for increasing the granularity of parallel programs. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 3, pp. 264–280, July 1991.

[68] D. L. Eager, J. Zahorjan, and E. D. Lazowska. Speedup cersus efficiency in parallel systems. *IEEE Transactions on Computers*, Vol. 38, No. 3, pp. 408–423, 1989.

[69] Arie Kaufman. *Volume visualization.* IEEE Computer Society Press tutorial. IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1991.

[70] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, pp. 29–37, May 1988.

[71] Lee Westover. Footprint evaluation for volume rendering. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90)*, Vol. 24(4), pp. 367–376. ACM SIGGRAPH, August 1990.

[72] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear–warp factorization of the viewing transformation. In Andrew Glassner, editor, *Computer Graphics (SIGGRAPH '94)*, Computer Graphics Proceedings, Annual Conference Series, pp. 451–458. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[73] T. S. Yoo, U. Neumann, H. Fuchs, S. Pizer, T. Cullip, J. Rhoades, and R. Whitaker. Direct visualization of volume data. *IEEE Computer Graphics and Applications*, pp. 63–71, July 1992.

[74] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In Arie Kaufman and Wolfgang Krueger, editors, *1994 Symposium on Volume Visualization*, pp. 91–98. ACM SIGGRAPH, October 1994. ISBN 0-89791-741-3.

[75] Allen Van Gelder and Kwansik Kim. Direct volume rendering with shading via three-dimensional textures. In *1996 Volume Visualization Symposium*, pp. 23–30. IEEE, October 1996. ISBN 0-89791-741-3.

[76] Hanspeter Pfister and Arie E. Kaufman. Cube-4 - A scalable architecture for real-time volume rendering. In *1996 Volume Visualization Symposium*, pp. 47–54. IEEE, October 1996. ISBN 0-89791-741-3.

[77] Peter Schröder and Gordon Stoll. Data parallel volume rendering as line drawing. *1992 Workshop on Volume Visualization*, pp. 25–32, 1992.

[78] William M. Hsu. Segmented ray casting for data parallel volume rendering. In Thomas Crockett, Charles Hansen, and Scott Whitman, editors, *ACM SIGGRAPH Symposium on Parallel Rendering*, pp. 6–14. ACM, November 1993.

[79] Kwan-Liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. A data distributed, parallel algorithm for ray-traced volume rendering. In Thomas Crockett, Charles Hansen, and Scott Whitman, editors, *ACM SIGGRAPH Symposium on Parallel Rendering*, pp. 15–22. ACM, November 1993. Color plates on page 105.

[80] V. Goel and A. Mukherjee. An optimal parallel algorithm for volume ray casting. *The Visual Computer*, Vol. 12, No. 1, pp. 26–39, 1996. ISSN 0178-2789.

[81] Philippe Lacroute. Real-time volume rendering on shared memory multiprocessors using the shear-warp factorization. In Stephen N. Spencer, editor, *Proceedings of the 1995 Parallel Renering Symposium*, pp. 15–22, New york, October 30–31 1995. ACM Press.

[82] Philippe Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorization. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, No. 3, pp. 218–231, September 1996.

[83] Minesh B. Amin, Ananth Grama, and Vineet Singh. Fast volume rendering using an efficient, scalable parallel formulation of the shear-warp algorithm. In Stephen N. Spencer, editor, *Proceedings of the 1995 Parallel Renering Symposium*, pp. 7–14, New york, October 30–31 1995. ACM Press.

[84] Tetu Hirai and Tsuyoshi Yamamoto. Accelerated composition for parallel volume rendering. *IEICE transactions on information and systems*, Vol. 81, No. 1, pp. 81–87, 1 1998.

[85] Patrick T. Gaughan and Sudhakar Yalamanchili. Adaptive routing protocols for hypercube interconnection networks. *Computer*, Vol. 26, No. 5, pp. 12–23, May 1993.

[86] T. Agerwala, J. Martin, J. Mirza, D. Sadler, D. Dias, and M. Snir. SP2 system architecture. *IBM Systems Journal*, Vol. 34, No. 2, pp. 152–184, 1995.

[87] S. W. White and S. Dhawan. POWER2: Next generation of the RISC System/6000 family. *IBM Journal of Research and Development*, Vol. 38, No. 5, pp. 493–502, September 1994.

[88] IBM. Ibm aix parallel environment, parallel programming subroutine reference, 2.0 edition, June 1994. 2.0 Edition.

[89] MPI Forum. MPI: A message-passing interface MPI forum. Technical Report CS/E 94-013, Department of Computer Science, Oregon Graduate Institute, March 94.

[90] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The Complete Reference. Volume 1, The MPI-1 Core.* Scientific and Engineering Computation. MIT Press, Cambridge, MA, USA, second edition, September 1998. See also volume 2 [91].

[91] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI: The Complete Reference. Volume 2, The MPI-2 Extensions.* Scientific and Engineering Computation. MIT Press, Cambridge, MA, USA, second edition, 1998. See also volume 1 [90].

[92] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, Vol. 1, No. 4, pp. 321–331, 1988.

[93] F. Leymarie and M. D. Levine. Tracking deformable objects in the plane using an active contour model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 6, pp. 617–634, June 1993.

[94] Shinji Umeyama. Contour extraction using complex autoregressive model. *IEICE Transactions on Information and Systems D-II, Information Processing*, Vol. 79, No. 3, pp. 358–364, 3 1996.

[95] Akira Amano, Yoshiyuki Sakaguchi, Michihiko Minoh, and Katso Ikeda. Snakes using a sample contour model. *IEICE Transactions on Information and Systems D-II, Information Processing*, Vol. 76, No. 6, pp. 1168–1176, 6 1993.

[96] A. P. Paplinski and J. F. Boyce. An implementation of the active contour method for noisy images using a local minimisation algorithm. Technical Report 95-1, Monash University, Department of Robotics and Digital Technology, 12 January 1995.

[97] Aboul-Ella HASSANIEN and Masayuki NAKAJIMA. Feature-specification algorithm based on snake model for facial image morphing. *IEICE transactions on information and systems*, Vol. 82, No. 2, pp. 439–446, 2 1999.

[98] L. D. Cohen. On active contour models and balloons. *Computer Vision, Graphics, and Image Processing. Image Understanding*, Vol. 53, No. 2, pp. 211–218, March 1991.

[99] Noboru YABUKI, Yoshitaka MATSUDA, Hiroyuki KIMURA, Yutaka FUKUI, and Shigehiko MIKI. Region extraction using color feature and active net model in color image. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, Vol. 82, No. 3, pp. 466–472, 3 1999.

[100] Ryo Furukawa, Masakazu Imai, and Takeshi Uno. Elastic contour model using spatio-temporal solid. *IEICE Transactions on Information and Systems D-II, Information Processing*, Vol. 79, No. 6, pp. 1054–1063, 6 1996.

[101] Ryo Furukawa, Kazumasa Imai, and Takechi Uno. Atmiq : active tubes on multiscale image queue. *IEICE Transactions on Information and Systems D-II, Information Processing*, Vol. 81, No. 4, pp. 611–622, 4 1998.

[102] Roman Ďurikovič, Kazufumi Kaneda, and Hideo Yamashita. Reconstructing a 3-D structure with multiple deformable solid primitives. *Computers & Graphics*, Vol. 21, No. 5, pp. 611–624, September 1997. ISSN 0097-8493.

[103] Amir A. Amini, Terry E. Weymouth, and Ramesh C. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-12, No. 9, pp. 855–867, September 1990.

[104] Amir A. Amini, Yasheng Chen, Rupert Curwen, Vaidy Mani, and Jean Sun. Coupled B-snake grids and constrained thin-plate splines for analysis of 2D tissue deforma-

tions from tagged MRI. *IEEE Transactions on Medical Imaging*, Vol. 17, No. 3, June 1998.

[105] D. J. Williams and M. Shah. A fast algorithm for active contours. In *The Third International Conference of Computer Vision*, pp. 592–595, 1990.

[106] Donna J. Williams and Shah Mubarak. A fast algorithm for active contours and curvature estimation. *Computer Vision, Graphics, and Image Processing. Image Understanding*, Vol. 55, No. 1, pp. 14–26, January 1992.

[107] C. T. Tsai, Y. N. Sun, and P. C. Chung. Minimising the energy of active contour model using a hopfield network. In *Iee Proceedings E*, pp. 297–303, 1993. Published as Iee Proceedings E, volume 140, number 6.

[108] Dong Joong KANG, Chang Yong KIM, Yang Seok SEO, and In So KWEON. A fast and stable method for detecting and tracking medical organs in mri sequences. *IEICE transactions on information and systems*, Vol. 82, No. 2, pp. 497–499, 2 1999.

[109] Kouta Fujimura, Naokazu Yokoya, and Kazuhiko Yamamoto. Motion analysis of nonrigid objects by active contour models using multiscale images. *IEICE Transactions on Information and Systems D-II, Information Processing*, Vol. 76, No. 2, pp. 382–390, 2 1993.

[110] Mark S. Nixon Steve R. Gunn. Global and local active contours for head boundary extraction. *International Journal of Computer Vision*, Vol. 30, No. 1, pp. 43–54, October 1998.

[111] Hiromi Kato, Ryo Furukawa, Masakazu Imai, and Takeshi Uno. Determination of weight coefficients of energy function of active tubes using neural networks. *IEICE Transactions on Information and Systems D-II, Information Processing*, Vol. 81, No. 9, pp. 1975–1982, 7 1997.

[112] Roman Ďurikovič, Kazufumi Kaneda, and Hideo Yamashita. Dynamic contour: a texture approach and contour operations. *The Visual Computer*, Vol. 11, No. 6, pp. 277–289, 1995. ISSN 0178-2789.

[113] Naonori Ueda, Kenji Mase, and Yasuhito Suenaga. A contour tracking method using elastic contour model and energy minimization approach. *IEICE Transactions on Information and Systems D-II, Information Processing*, Vol. 75, No. 1, pp. 111–120, 1 1992.

[114] Masahiro Hashimoto, Hirotsugu Kinoshita, and Yoshinori Sakai. An object extraction method using sampled active contour model. *IEICE Transactions on Information and Systems D-II, Information Processing*, Vol. 77, No. 11, pp. 2171–2178, 11 1994.

[115] Michael Hoch and Peter C. Litwinowicz. A semi-automatic system for edge tracking with snakes. *The Visual Computer*, Vol. 12, No. 2, pp. 75–83, 1996. ISSN 0178-2789.

[116] F. W. Burton. Speculative computation, parallelism and functional programming. *IEEE Transactions on Computers*, Vol. 34, No. 12, pp. 1190–1193, 1985.

[117] P. V. R. Murthy and V. Rajaraman. Implementation of speculative parallelism in functional languages. *IEEE Transactions on Computers*, Vol. 5, No. 11, pp. 1197–1205, 1994.

Autholized Paper List

Reviewed Papers

- Hiroyuki Kitajima, Mitsuhiro Nakaizumi, Hong Shen, Hiroaki Kobayashi, and Tadao Nakamura, "Task Scheduling Strategies and Their Locality Evaluation of Memory References on a Parallel Graph Reduction System," IPSJ Transactions, Vol.37, No.11, pp.2020–2029, 1996.11. (in Japanese) [Chapter 3]
- Kentaro Sano, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura, "Data-Parallel Volume Rendering with Adaptive Volume Subdivision," IEICE Transactions on Information and Systems, Vol.E83-D, No.1, pp.80–89, 2000.1. [Chapter 4]
- Hiroyuki Kitajima, Yoshiyuki Kaeriyama, Hiroaki Kobayashi, and Tadao Nakamura, "An Active Contour Model Using Local Shape Information of a Region-of-Interest and its Parallel Processing," submitted to IPSJ Transactions on High Performance Computing Systems. (in Japanese) [Chapter 5]

International Conference Papers

- Hong Shen, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura, "A Hierarchical Parallel Reduction System for the Functional Language FL," Proceedings of the High Performance Computing Conference '94(HPCC '94), pp.270-279, 1994.9, (Singapore). [Chapter 3]
- Hong Shen, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura, "Task Scheduling with Locality Consideration for a Clustered Parallel FL Reduction Sys-

tem," Proceedings of the First Aizu International Symposium on Parallel Algorithms/Architecture Synthesis(pAs '95), pp.234–240, 1995.3, (Aizu University). [Chapter 3]

- Kentaro Sano, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura, "Parallel Processing of the Shear-Warp Factorization with the Binary-Swap Method on a Distributed-Memory Multiprocessor System," Proceedings of the Third Parallel Rendering Symposium '97 (PRS '97, in conjunction with IEEE Visualization '97), pp.87–94, p.118, 1997.10, (Phoenix, U.S.A.). [Chapter 4]

Lecture Notes (in Japanese)

- Hong Shen, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura, "Performance Evaluation of a Hierarchical Parallel Reduction System for FL," Functional Programming II JSSST'94, Lecture Note/Software 10, Masato Takeichi (edu.), Kindaikagakusya Co. Ltd., pp.159–174, 1994.11, (Kyoto). [Chapter 3]

- Hiroyuki Kitajima, Hong Shen, Masayuki Katahira, Hiroaki Kobayashi, and Tadao Nakamura, "Performance Studies of the FL Hierarchical Parallel Reduction System," Lecture Notes of IPSJ (95-HPC-56), Vol.56, No.1, pp.1–8, 1995.6, (SIG-HPC, IBM Japan Ltd., Hakozaki, Tokyo). [Chapter 3]

- Kentaro Sano, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura, "Adaptive Volume-Subdivision for Efficient Data-Parallel Volume Rendering," Lecture Notes of IPSJ (98-HPC-73), Vol.73, No.2, pp.7–12, 1998.10, (SIG-HPC, Tohoku University). [Chapter 4]

National Convention Papers (in Japanese)

- Hiroyuki Kitajima, Hong Shen, Hiroaki Kobayashi, and Tadao Nakamura, "Optimization of Scheduling Strategies for a Pipelined Reduction Machine," The 29th National Convention Record of Tohoku Branch of the Japan Society of Mechanical Engineers, No.941-1, pp.118–120, 1994.3, (Tohoku University). [Chapter 3]

- Mitsuhiro Nakaizumi, Hiroyuki Kitajima, Hong Shen, Hiroaki Kobayashi, and Tadao Nakamura, "A Memory System of the Parallel Reduction Machine," 1996 Tohoku-Section Joint Convention Record of Institute of Electrical and Information Engineers, Japan, p.101, 1996.8, (Tohoku Gakuin University). [Chapter 3]

- Yoshinori Kimura, Kentaro Sano, Hiroyuki Kitajima, Hiroaki Kobayashi, and Tadao Nakamura, "A Study of Virtual Reality System with Volume Rendering," 1998 Tohoku-Section Joint Convention Record of Institute of Electrical and Information Engineers, Japan, p.236, 1998.8, (Tohoku University). [Chapter 4]

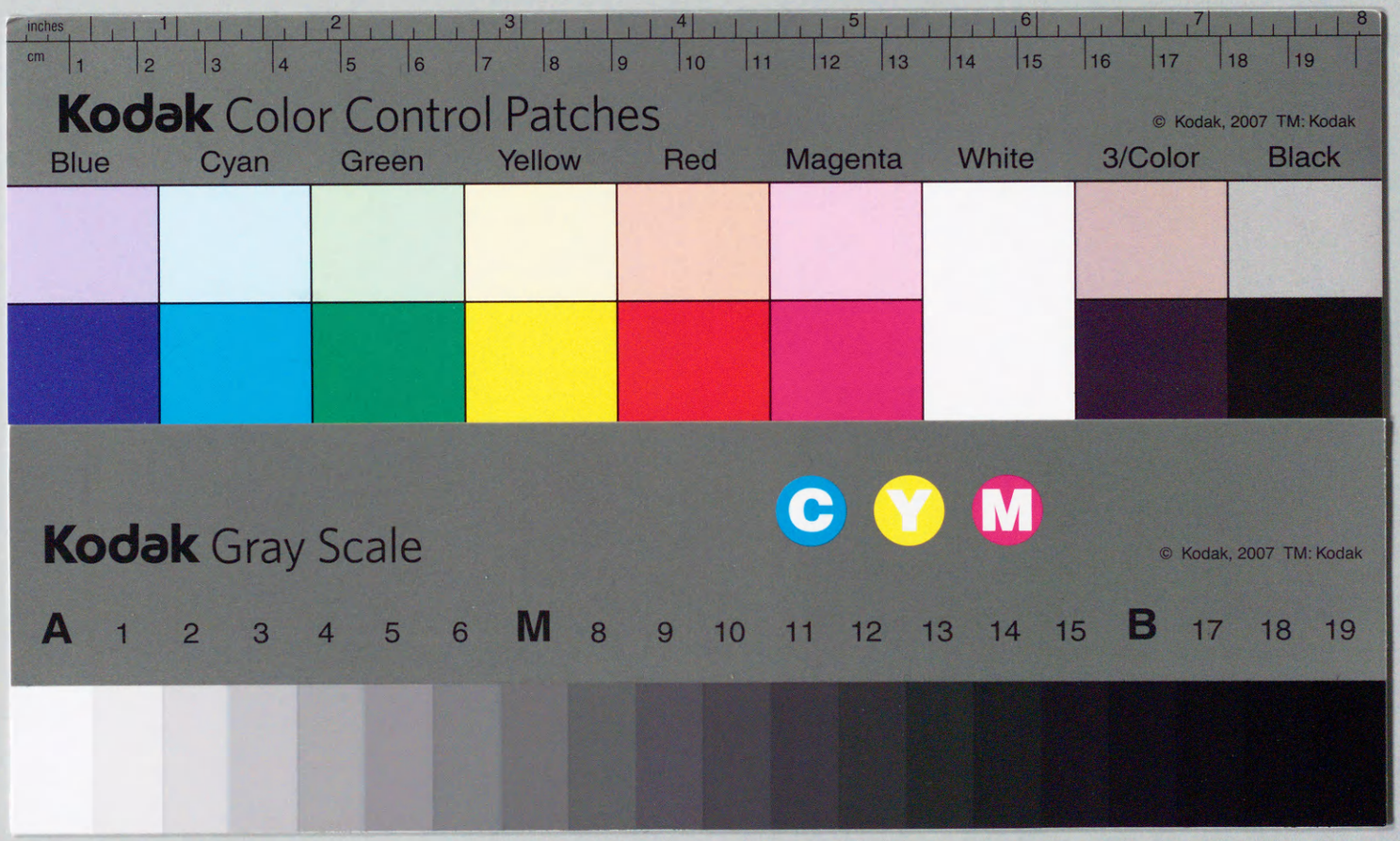
- Hiroyuki Kitajima, Yoshiyuki Kaeriyama, Hiroaki Kobayashi, and Tadao Nakamura, "An Active Contour Model using Regions-of-Interest Shape Information," The 59th National Convention Record of Information Processing Society of Japan, Vol.2, pp.257–258, 1999.9, (Iwate Prefectural University). [Chapter 5]

Thesis of Master Degree

- Hiroyuki Kitajima, "Performance Evaluation of a Functional Architecture," Department of Computer and Mathematical Sciences, Graduate School of Information Sciences, Tohoku University, 1995.2.

Thesis of Bachelor Degree

- Hiroyuki Kitajima, "Pipelined Reduction Engine," Department of Mechanical Engineering, School of Engineering, Tohoku University, 1993.3.


inches
cm
Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
Kodak Gray Scale
C
Y
M
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19